PARALLEL DIGITAL SIGNAL PROCESSING ON A NETWORK OF PERSONAL COMPUTERS CASE STUDY: SPACE-TIME ADAPTIVE PROCESSING

THESIS

Fernando Silva, Captain, BAF

AFIT/GCS/ENG/99J-01

19990629 050

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

DTIC QUALITY INSPECTED 4

AFIT/GCS/ENG/99J-01

PARALLEL DIGITAL SIGNAL PROCESSING ON A NETWORK OF PERSONAL COMPUTERS CASE STUDY: SPACE-TIME ADAPTIVE PROCESSING

THESIS

Fernando Silva, Captain, BAF

AFIT/GCS/ENG/99J-01

Approved for public release; distribution unlimited

AFIT/GCS/ENG/99J-01

PARALLEL DIGITAL SIGNAL PROCESSING ON A NETWORK OF PERSONAL COMPUTERS CASE STUDY: SPACE-TIME ADAPTIVE PROCESSING

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Science

Fernando Silva, B.S.S.A

Captain, Brazilian Air Force

June, 1999

Approved for public release; distribution unlimited

AFIT/GCS/ENG/99J-01

PARALLEL DIGITAL SIGNAL PROCESSING ON A NETWORK OF PERSONAL COMPUTERS CASE STUDY: SPACE-TIME ADAPTIVE PROCESSING

THESIS

Fernando Silva, Captain, BAF

Approved:

[illegible] 20 MAY 1999

Gary B. Lamont, Ph.D. (Chairman) date

[illegible] 20 May 99

Major Richard A. Raines, Ph.D. (Member) date

[illegible] 20 May 99

Steven C. Gustafson, Ph.D. (Member) date

Acknowledgements

I would like to thank everyone who directly or indirectly played a part in this research effort. I thank my Thesis Committee, and specially my Thesis Advisor, Dr. Gary B. Lamont for all the foresight and freedom provided throughout my thesis and course work.

Many thanks go to my fellow graduate students from the Parallel Laboratory. The help and camaraderie provided by Capt. Christopher Bohn, Capt. Karl Deerman, Capt. Lonnie Hammack, and Lt. Alexandre Valente is most appreciated.

Finally, I would like to thank my wife, Rosane, for all her help, support, and encouragement.

Table of Contents

List of Figures

List of Tables

AFIT/GCS/ENG/99J-01

Abstract

Network-based parallel computing using personal computers is currently a popular choice for concurrent scientific computing. This work evaluates the capabilities and the performance of the AFIT Bimodal Cluster (ABC) - a heterogeneous cluster of PCs connected by switched fast Ethernet and using MPICH 1.1 for interprocess communication - for parallel digital signal processing using Space-Time Adaptive Processing (STAP) as the case study. The MITRE RT_STAP Benchmark version 1.1 is ported and executed on the ABC, as well as on a cluster of six Sun SPARC workstations connected by a Myrinet network (the AFIT NOW), and on a IBM SP for comparison. Modifications to the RT_STAP benchmark source code are done to accommodate the BLAS routines obtained from the ASCI Red project and the FFTPACK from the Netlib repository. Comparative performance analysis of the original and modified versions of the benchmarks executed on the ABC running the LINUX OS is performed, and shows improvements in the sustained Gflop/sec rates. Inter-platform comparative analysis demonstrates ABC's superior computation rates, but also reveals limited machine scalability as a result of severe communication overheads imposed by RT_STAP cornerturn operations. Analysis of experimental data indicates that ABC outperforms AFIT NOW but needs interconnection network improvements to be globally competitive to MPPs such as the IBM SP.

PARALLEL DIGITAL SIGNAL PROCESSING ON A NETWORK OF PCs

CASE STUDY: SPACE-TIME ADAPTIVE PROCESSING

I. Introduction

We have witnessed an extremely accelerated growth in the performance and capability of digital signal processing systems for the last two decades. The main theme for this dramatic story is the advance of the underlying VLSI technology, which allows larger and larger numbers of computational components to fit in a chip, and associated clock rates to increase. In this scenario, computer architecture (hardware and software) technology and applications evolve together and have very strong interactions. Specifically, one of the main characteristics of signal processing applications is the strong demand for computational cycles, which often supersedes the capacity offered by state-of-the-art microprocessors. Also, the embedded nature of most of these applications has also led to strong constraints on cost, size, and power. As a result, signal processing designs lead the way in the development of special-purpose processors. More recently, these designs have influenced the development of new capabilities in general-purpose processors as well.[Rabaey98].

The concept of adaptive signal processing is not new. Since the early 1970's effective algorithms have been available for adaptively minimizing the effects of external signal interference on airborne radar operations on the basis of its spatial and spectral characteristics, but they received little attention until the mid 1980's. The reason for this gap was twofold: the computational throughput needed to implement the technique was well beyond the capabilities of the airborne processors available, and the engineering requirements for electronically-steered array antennas (ESA), receivers, and A/D converters were not completely affordable in the 1970's [Stimson98].

1.1 Platforms

Advances in hardware capability enable new application functionality, which grows in significance and places greater demands on the architecture. This cycle drives the tremendous ongoing research, engineering, and manufacturing effort underlying the increase in microprocessor performance, as can be seen in Figure 1. As advances in technology determine what is possible, computer architects translate the potential of the technology into performance and capability mainly through the use of the principles of parallelism and locality [Kumar94][Dongarra98]. Whenever multiple operations are performed concurrently, the number of cycles required to execute the task is reduced.

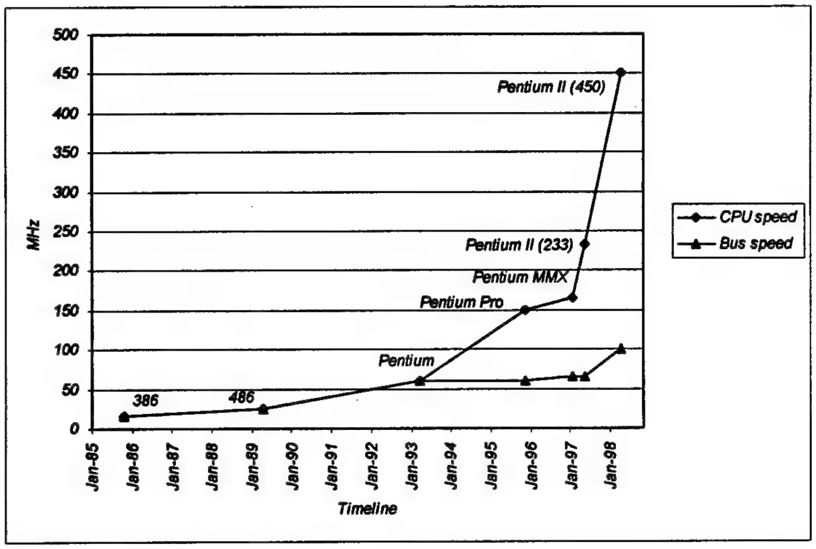


Figure 1 – Improvement in CPU and Bus speed for some Intel© microprocessors.

Also, whenever data references are performed close to the processor element, the latency of accessing deeper levels of the memory hierarchy is avoided, and the number of cycles required to execute the task is reduced as well. All contemporary microprocessors realize highly parallel functionality by employing pipelining, superscalar, or VLIW techniques, executing several instructions in the same clock cycle, and reordering instructions within the limits of the inherent dependencies to reduce the costs of communication with hardware components external to the processor [Patterson98].

The direct reliance on increasing levels of performance is well established in a number of endeavors, but it is most apparent in the field of computational science and engineering. Today, to obtain performance significantly greater than the state-of-the-art microprocessor, the primary option is the use of multiple processor systems (supercomputers), while the most demanding applications are written as parallel programs [Culler98]. Thus, parallel computer architectures and distributed applications are subject to the strongest demands for greater performance.

Figure 2 summarizes the findings of the Committee on Physical, Mathematical, and Engineering Sciences of the Federal Office of Science and Technology Policy [OST93]. It indicates the computational rate and the storage capacity required to attack a number of important science and engineering problems. The Grand Challenge problems scientists face today are awesome in their computational requirements, and even with dramatic increases in processor performance, very large parallel architectures are needed to address these problems in the near future [Culler98]. A recent and comprehensive list containing these challenging problems can be found in [Chorafas97:192].

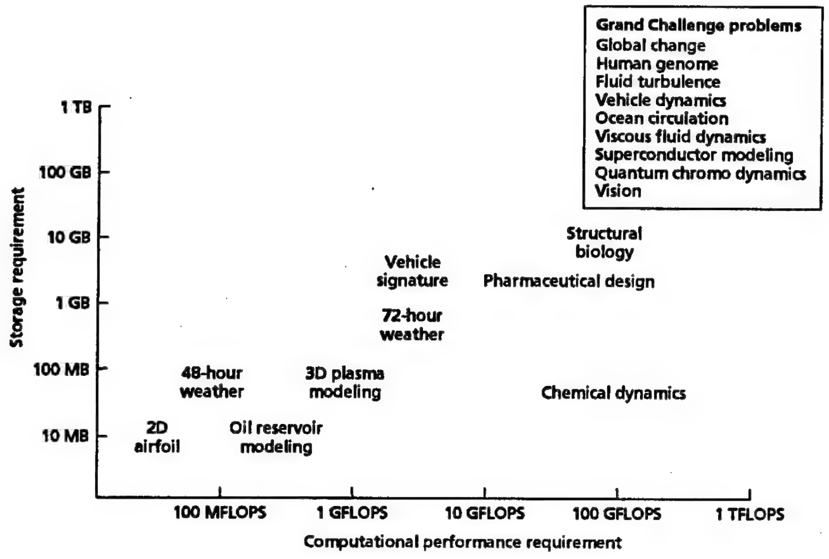


Figure 2 - Grand Challenge problems.

The microprocessor based supercomputers provided initially about a hundred processors, increasing to roughly a thousand from 1990 onward. These massively parallel processors (MPP) – machines constructed as a large collection of workstation-class nodes connected by a dedicated scalable low latency network – have tracked the microprocessor advance, with typically a lag of one to two years behind the leading microprocessor-based workstation or personal computer [Anderson95]. These MPP computers are asynchronous MIMD machines [Flynn72]. Along with the use of commodity components (microprocessors, disks, memory chips), they also employ a distributed memory model driven by larger processor counts, and by the rapid increase in processor's performance and associated increase in memory bandwidth requirements. MPPs are also characterized by the fact that the physically separate memories are viewed as multiple private address spaces that are logically disjoint and cannot be referenced by a remote processor. As a consequence, communication between processors is done by explicit message passing [Hennessy96].

Despite being the popular choice for concurrent computing, MPP architectures have two major disadvantages: high acquisition and maintenance costs, and an engineering lag time that associated with the rapid increase in the performance of commodity components – especially microprocessors – end up costing more than a factor of two in the bottom line computational performance for a two-year lag [Anderson95]. In other words, the desire for high-performance spares no cost in achieving its goal, while the time needed for developing a high-speed, low-latency interconnection network, operating system, communication library software, and supporting hardware ends up affecting the time-to-market for the system.

An alternative platform for high-performance scientific concurrent computing is the commodity computer cluster. Traditionally, this collection of complete computers with a dedicated interconnect have been used to serve multiprogramming workloads and to provide higher availability [Pfister98]. However, the technology breakthrough that presents the potential of clusters to take on an important role in large scale parallel computing is a scalable, low-latency interconnect, similar in quality to that available in parallel machines, but deployed like a LAN. The introduction of such standardized communication facilities such as ATM, FDDI, 100 Base-T Ethernet, Gigabit Ethernet, FibreChannel, and Myrinet, raised communication bandwidth from 10 Mbits/sec up to the order of Gigabits/sec [Lauria98].

There are, nevertheless, some technical challenges inherent in realizing the opportunities of clusters as cost-effective platforms for concurrent computing. The philosophy of employing commercial-of-the-shelf (COTS) hardware components and software helps the transfer of new ideas and technology and simplifies assembling and maintenance at low cost. However, the absence of specialized software to provide single system image, and efficient task scheduling inhibits the potential that clusters have for computation. Another important issue is the availability of a fast messaging layer that does not impose large software overheads for communication and capitalizes on the full power of the interconnection networks. These are two software related problems that must be solved before clusters may be considered globally competitive to MPPs [Pfister98]. Thus, a significant amount of research is being conducted in order to improve messaging layers and develop new operating system layers [Culler96][Lauria98].

The Air Force Institute of Technology (AFIT) has recently initiated the building of a cluster of Intel Pentium II© personal computers networked by a 100 Base-T switched Ethernet using the Message Passing Interface standard (MPI) as the communication mechanism. This network-based platform, called *AFIT Bimodal Cluster (ABC)*, operates under both Linux 2.0.33 and Windows NT 4.0 operating systems, and has been used for research on signal processing, distributed databases, computational fluid dynamics, and evolutionary computation. The current configuration comprises 12 personal computers of different clock speeds, from 200 to 450 MHz. Plans for expansion include the acquisition of Symmetric Multiprocessing (SMP) platforms. A more detailed description of the ABC is provided in the Appendix B.

1.2 Objective

The objective of this thesis effort is to investigate the capabilities and the performance of the ABC as a platform for parallel digital signal processing, specifically applying *Space-Time Adaptive Processing (STAP)* [Ward94] as the object of a *case study* [Zelkowitz98]. The reasons for this choice are the following: (a) STAP is a computationally demanding technique for mitigating clutter and jamming as seen by airborne radar, and incorporates several important numerical algorithms such as the computation of the solution of linear systems of equations, discrete Fourier transforms, and matrix factorizations, which are of interest for an investigation of ABC as a platform for parallel scientific computing; (b) STAP incorporates different signal processing tasks such as pulse compression, Doppler processing, and adaptive processing, therefore comprising a representative case study with regard to signal processing in general; (c) the system is guaranteed to be sufficiently stressed [Barr95], because STAP processing requirements are challenging even for the state-of-the-art multiprocessors / multicomputers available today.

Our hope is that the conclusions and insights provided by this research effort can be beneficial to the high-performance and signal processing communities both in the United States and Brazilian Air Forces, since it represents an opportunity to investigate the use of general-purpose microprocessors and network-based parallel computing in the digital signal processing realm. Specifically, this thesis effort is the first attempt to address the computational capabilities and the cost/performance ratio provided by a cluster of personal computers when applied to STAP.

1.3 Approach

Investigating the use of the ABC for STAP requires that three basic steps be accomplished: (a) select and obtain a STAP implementation; (b) port it into the cluster environment; (c) check for effectiveness and performance of the results. The last step is explored further by modifying the original source code in order to allow it to increase its performance on the host platform while maintaining portability of the original implementation.

A comparison between the results obtained from two different implementations then provides necessary information for evaluating the effects of the modifications applied to the original code, and for evaluating the performance of the modified STAP implementation while running on the cluster. Additionally, two other platforms are used as hosts for the STAP implementation: The *AFIT NOW* (a cluster of six Sun© Sparc workstations connected by a crossbar Myrinet© switch) and the *IBM SP Multicomputer* located at the Aeronautical Systems Center Major Shared Resource Center – ASC MSRC, Wright-Patterson AFB. The results obtained from STAP processing on these platforms provide additional insight on the characteristics of the application and on the capabilities and scalability of the ABC platform. Finally, reasoning upon experimentation and results of comparative analysis, conclusions are derived concerning the performance of the ABC itself in regard to STAP.

1.4 Organization

This thesis document is organized around six chapters. **Chapter II** provides the background information necessary to understand the general topics introduced in the first chapter. Thus, it includes (1) information about the role of parallel computing and parallel architectures in the effort to solve computationally intensive problems in the area of signal processing, (2) some distinctive characteristics of MPPs, SMPs, and clusters for concurrent computing, (3) a literature review on space-time adaptive processing algorithms and software implementations, (4) basic radar operation, and (5) ABC, AFIT

NOW and IBM SP hardware / software configurations. **Chapter III** reports the methodology that guided this thesis work, and it is organized around two sections. The first part details the actions taken to achieve the objectives of the thesis research. Thus, it contains a description of the aims, order, and nature of the work performed . The second part deals with the problem domain / algorithm domain integration, and contains a detailed description of the selected STAP implementations and associated complexities. **Chapter IV** refers to the experimental framework, including the design of the experiments performed according to the level of observation and objectives of the thesis effort, as well as the corresponding performance metrics. This section also lists our assumptions. **Chapter V** reports and analyzes the performance of selected STAP implementations before and after the source code modifications, as well as the performance of the ABC while running STAP. Results obtained on the alternative platforms are used for comparisons. Explanations for the performance differences are provided based on the results of experimentation and statistical analysis of the data collected. **Chapter VI** reports the conclusions of the investigation and allows quantitative and qualitative answers to the objectives of this thesis effort. Recommendations for future research based on the experience with this study are also provided.

The reader is assumed to have a basic understanding of concepts related to parallel computer architectures and concurrent message-passing programming.

II. Background

2.1 Introduction

This chapter provides the background information necessary to understand the general topics introduced in the first chapter. Therefore, it includes information about the role of parallel computing and parallel architectures in the effort to solve computationally intensive problems in the area of signal processing; some distinctive characteristics of MPPs, SMPs, and clusters for concurrent computing; a literature review on space-time adaptive processing algorithms and basic radar operation (problem domain description); and the ABC, NOW, and IBM SP hardware / software configurations.

2.2 Motivation for STAP

Modern airborne radar platforms are required to provide long-range detection of smaller and smaller targets in the presence of severe interference from both natural and artificial sources. This detection of targets is often performed over land, where ground clutter can be very high [Skolnik62], and in the presence of electronic countermeasures such as jamming [Skolnik62, Toomay89]. These radar platforms must have the capability to nullify both clutter and jamming to below the ambient noise level.

The suppression of jamming and clutter has posed a problem to radar engineers since the beginning of radar. Over the years, many techniques have been developed to try and eliminate jamming and clutter; however, the problem is difficult because it is dependent on a number of different inter-related variables. A potential target may be obscured not only by the mainlobe clutter (i.e., the clutter that originates from the same angle as the target) but also by the sidelobe clutter (i.e., the clutter that comes from different angles but has the same Doppler frequency) [Toomay89].

A typical airborne radar scenario is sketched below in Figure 3. The aircraft motion spreads the clutter in Doppler, with ground clutter coming from azimuths behind the aircraft having negative relative velocity and ground clutter ahead of the platform having positive Doppler. In its entirety, the ground clutter lies on the clutter ridge, when viewed in azimuth and Doppler. For this example, the radar is looking broadside (normal to velocity vector) so the mainbeam clutter is centered at zero Doppler. A small target, coming from the mainbeam direction may be obscured by both mainlobe clutter at the same angle but a different Doppler, and by sidelobe clutter that has different angle but the same Doppler as the target.

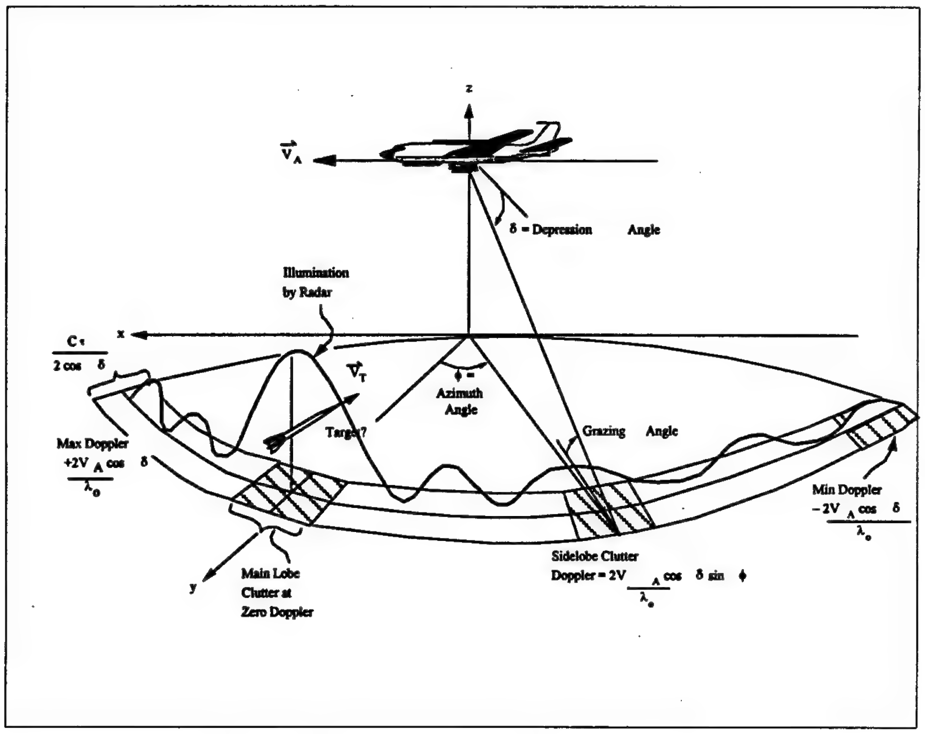

z
$\vec{V}_A$
δ = Depression Angle
Illumination by Radar
x
$\frac{C\tau}{2\cos\delta}$
$\vec{V}_T$
φ = Azimuth Angle
Target?
Grazing Angle
Max Doppler $\frac{+2V_A\cos\delta}{\lambda_o}$
Min Doppler $\frac{-2V_A\cos\delta}{\lambda_o}$
Sidelobe Clutter Doppler = $\frac{2V_A}{\lambda_o}\cos\delta\sin\phi$
y
Main Lobe Clutter at Zero Doppler

Figure 3 – Airborne Radar Scenario.

Displaced-phase-center-antenna (DPCA) processing was developed to address the problem of clutter in airborne radar platforms [Staud90]. The effects of jamming on radar systems can often be successfully cancelled by adaptive array processing techniques [Maill93]. The above two techniques – DPCA and adaptive array processing – individually provide a partial solution to the problem of clutter and jamming, respectively. These two techniques have been effectively combined in a technique known as Space-Time Adaptive Processing (STAP), which can be viewed as a generalization of DPCA processing [Ward94]. STAP simultaneously and adaptively combines the signals received on multiple elements of an antenna array – the spatial domain – and from multiple pulse repetition periods – the temporal domain.

STAP offers the potential to improve airborne radar performance in several areas. STAP algorithms can provide improved target detection in the presence of interference through the adaptive nulling of both ground clutter and signal jamming [Ward94]. It can improve low velocity target detection through better mainlobe clutter suppression. It can also be used to detect small targets, which would otherwise be obscured by the presence of sidelobe clutter. STAP also provides a capability to cancel nonstationary interference. *Thus, STAP combines both spatial and temporal adaptive processing techniques to cancel out the clutter and interference contained in the radar signals received by an airborne antenna array.*

Another significant feature of STAP is that it can improve the performance of the antenna array while requiring little or no modification to the basic radar design. However, the computational complexity associated with STAP is generally very high [Ward94:77];

an extremely large amount of data needs to be processed in real-time. This in turn requires a large computational throughput.

2.3 Digital Signal Processors (DSP)

As a result of the changes in governmental procurement methodology and also in military cost reductions, the Department of Defense (DoD) is moving towards commercial-off-the-shelf (COTS) products for the design and deployment of military systems. There are a number of embedded military applications such as airborne target recognition systems, undersea sonar platforms, ground processing stations, and command and control systems in which non-commercial resources are being abandoned. In particular, COTS parallel processing systems are replacing custom embedded military sonar and radar systems on ships and airborne aircraft [Rowe96].

In contrast to contemporary non-commercial products that involve costly custom engineering, ideally, COTS products offer lower cost hardware, faster development that reduces program lifecycle costs, and higher reliability while adhering to strict size, weight, and power (SWAP) requirements of many military applications. These characteristics of commercial products are achievable simply because of volume production and compatibility with a wide range of applications. Furthermore, the practice of purchasing COTS equipment creates a competitive market that stimulates both technological advancement and decreased costs [Rowe96].

Digital signal processing is one of the core technologies central to the operation of military-based radar systems. Digital signal processing is the application of mathematical operations on a digitally represented sequence of samples from an analog signal. Since their emergence in the late 1980s, DSPs have experienced tremendous growth rates in areas of signal processing due to reductions in costs, advances in DSP architectures, and improvements in development tools [Rabaey98]. Simply stated, a DSP is a special purpose microprocessor similar to a traditional microprocessor that is optimized to perform mathematical operations such as multiplications, additions, and subtractions with greater efficiency. In addition to their increased performance for a class of computations, DSPs are generally less expensive than general-purpose microprocessors. Table 1 summarizes the characteristics of a class of floating-point DSP.

Texas Instruments TMS320C67X Family of floating point DSP	
Cycle time (nsec)	6
Clock rate (MHz)	167
Performance (Gflop/sec)	1
Architecture	Load-store, 32 32-bit registers, 8 FU (5 fp and 3 int)
Cache	512 Kbit data/instruction each
Nominal voltage (v)	1.8 to 3.3
Price (U$)	109 to 233

Table 1 – Features of a typical class of digital signal processors.

Given the large applicability of embedded signal processing systems, the landscape in terms of different design processes for such systems is complex and shows important relationships involving the hardware/software techniques used. Usually, total cost and time-to-market for those systems are highly sensitive to those relationships, and key architectural attributes considered are computation/communication performance, topology, software (application and control), and interfaces. An illustrative example is the work presented in [James95], where parametric cost-estimation techniques were used to examine the effects of memory and processor constraints on the software development costs of a synthetic aperture radar (SAR) processor. The results showed how development costs can be dominated by software, and that a greater investment in hardware resources (memory and processors) can substantially reduce overall system development costs. Table 2 shows the comparison of costs and development time between minimum hardware cost and minimum total cost scenarios.

Cost	Hardware costs	Software costs	Total cost	Development time
Minimum hardware cost	$281,000	2,360,000	2,640,000	32 months
Minimum total cost	$432,000	$911,000	1,343,200	28 months

Table 2 – Comparison of costs for the work in [James95].

The net result was a superior product at a half the cost of the minimum hardware product, mainly because the programming and tuning efforts employed to obtain high levels of processor/memory utilization was alleviated (being kept to 50% maximum) by the availability of more hardware resources. The tendency of using COTS for building signal processing systems facilitates this process, by allowing the hardware costs to be even lower. In fact, contemporary signal processing systems have less than 20% hardware cost on the average, as opposed to 80% cost for the software component [DeBar98].

2.4 Parallel Signal Processing

Classical signal processing algorithms are characterized by the need for high-performance and involve repetitive, numerically-intensive tasks, which are the targets for DSP technology. However, processing speeds of a single DSP are often insufficient to satisfy the computation demand of military-based signal processing applications. For such real-time signal processing applications, parallel processing is required to meet the necessary performance requirements [Choudhary97]. Typically, parallel processing is organized around the set of distinct tasks that comprise the original signal processing application, such as filtering, convolution, pulse compression, etc., and the set of available processors used can be assigned to deal with each phase in a dataflow fashion (in this case the output for one task serves as the input to the next task, and the input data for a given task is divided among the processors), or they can be divided in subsets, each one responsible for processing a subset of the total number of tasks that comprise the application in a pipeline fashion. Generally, and depending on the nature of the task, when more than one processor is assigned to it, we can divide the input data between the processors for sequential processing, or the processors can all work in parallel on the totality of the input (the task is parallelized), or both options are used to accomplish the task.

Considerable progress has been made in using COTS embedded high performance computers to implement signal processing for real-time applications that in the past would have required the development of special purpose processors [Games98]. A recent computer platform evaluation effort cited in [Games98] chose an SGI© processor for a ground-based application and a Mercury© processor in two configurations for a more strict embedded airborne application. Information on these machines is shown in Table 3.

Computer	Number of processors	Processor	Gflops (peak)
Silicon Graphics Origin 2000	128	MIPS R10000	50
Mercury MP 420	840 + 16	SHARC + PowerPC respectively	100
Mercury MP 420	296	PowerPC	59

Table 3 – Two COTS embedded high-performance computers used for parallel signal processing.

The fact that parallel signal processing applications, once developed, are likely to be used for a long time, and also require the highest performance, suggests the use of message passing as the communication mechanism between processors in order to allow the generation of programs with better efficiency [Kumar94:120] and portability. In that sense, both MPPs and clusters have the message passing mechanism as their native communication model [Hwan96]. Table 4 summarizes several attributes for comparison of MPPs and clusters for adaptive signal processing.

Another reason for high levels of parallel program efficiency is that it has a direct impact on size, weight, power, reliability, and cost of the system, which are important factors considering airborne radar platforms. In this sense, both fine and coarse-grained implementations of STAP algorithms have been considered [Samson96].

Application Attributes	**Massively Parallel Processors – MPPs**	**Clusters of Workstations/PCs**
Number of Nodes	Hundreds to thousands	Tens to hundreds
Reported Performance (Gflops)	Tens to hundreds	Less than ten
Task Granularity	Dedicated single-tasking per node	Multitasking or multiprocessing per node
Internode Communication and Security	Proprietary network and enclosed security	Often standard
Node Operating System	Homogeneous microkernel	Could be heterogeneous, often homogeneous; complete Unix/Linux/Windows NT
Strength and Potential	High throughput with higher memory and I/O bandwidth	Higher availability with easy access of large-scale database managers
Application Software	Signal processing libraries exist and portable	**Untested for signal processing applications**
Shortcomings and Open Problems	Expensive and lack of real-time OS support	Heavy communication overhead and lack of single system image

Table 4 – Comparison of MPPs and NOWs for Adaptive Signal Processing [Hwan96].

2.5 Radar Fundamentals

The fundamental purpose of radar is to detect the presence of an object of interest and provide information concerning that object's range, velocity, angular coordinates, size, and other parameters [Rihac69]. An elementary form of radar consists of a transmitting antenna and a receiving antenna. Radar operates by radiating electromagnetic (EM) energy, oscillating at a predetermined frequency, and duration, into free space through the transmitting antenna. In general, the radar antenna forms a beam of EM energy that concentrates the EM wave into a given direction [Eaves87]. By effectively rotating and pointing the antenna, the transmitted radar signal can be directed to a desired angular coordinate.

A portion of the radar's transmitted energy is intercepted by an object located in the path of the transmitted beam and is scattered in all directions depending on the target's physical characteristics. In general, some of the transmitted energy is reflected back in the direction of the radar. This retro-reflected energy is referred to as backscatter [Eaves87]. The radar antenna receives a portion of the backscattered wave, or echo return. The echo returns, which are gathered by a set of sensors, are sampled, and the resulting data is processed to identify targets and parameter estimation.

The distance to the target is determined by measuring the time taken for the radar signal to travel to the target and back. Furthermore, the angular position of the target may be determined by the arrival direction of the backscattered wave. If relative motion exists between the target and radar, the shift in the carrier frequency of the reflected wave, also known as the Doppler effect, is a measure of the target's relative velocity and may be used to distinguish moving targets from stationary objects [Toomay89].

The basic role of the radar antenna is to act as a transducer between the free-space propagation and guided-wave propagation of the EM wave [Skolnik90]. The specific function of the antenna during transmission is to concentrate the radiated energy into a shape beam directive that illuminates targets in a desired direction. During reception, the antenna collects the energy from the reflected echo returns. Several varieties of radar antennas have been used in radar systems, such as the parabolic reflectors and the planar array antennas [Stimson98:95]. The type of radar antenna selected for a certain application depends not only on the electrical and mechanical requirements dictated by the radar design specifications but also on its application. In airborne-radar applications, radar antennas must generate beams with shape directive patterns that can be scanned.

The properties offered by antenna arrays are quite appealing to airborne radar systems. Antenna arrays consist of multiple stationary elements, which are fed coherently, and use phase or time-delay control at each element to scan a beam to given angles in space [Morris88]. The primary reason for using radar arrays is to produce a directive beam that can be repositioned electronically. An electronically steerable antenna array, whose beam steering is inertialess, is drastically more cost effective when the mission requires surveying large solid angles while tracking a large number of targets [Morris88]. Additionally, arrays are sometimes used in place of fixed aperture antennas because the multiplicity of elements allows a more precise control of the radiating pattern.

The purpose of moving-target indication (MTI) radar is to reject signal returns from stationary or unwanted slow-moving targets, such as buildings, hills, trees, sea, rain, and snow, and retain detection information on moving targets such as aircraft and missiles [Rowe96]. The term Doppler radar refers to any radar capable of measuring the shift between the transmitted frequency and the frequency of reflections received from possible targets [Taylor48]. Relative motion between a signal source and a receiver creates a Doppler shift of the source frequency. When a radar system intercepts a moving object that has a radial velocity component relative to the radar, the reflected signal's frequency is shifted.

The Doppler effect is a shift in the frequency of a wave radiated, reflected, or received by an object in motion [Stimson98:189]. As illustrated in Figure 4, a wave radiated from a point source is compressed in the direction of motion and is spread out in the opposite direction. Only at right angles to the motion is the wave unaffected. Since frequency is inversely proportional to wavelength, the more compressed the wave is, the higher its frequency is, and vice versa.

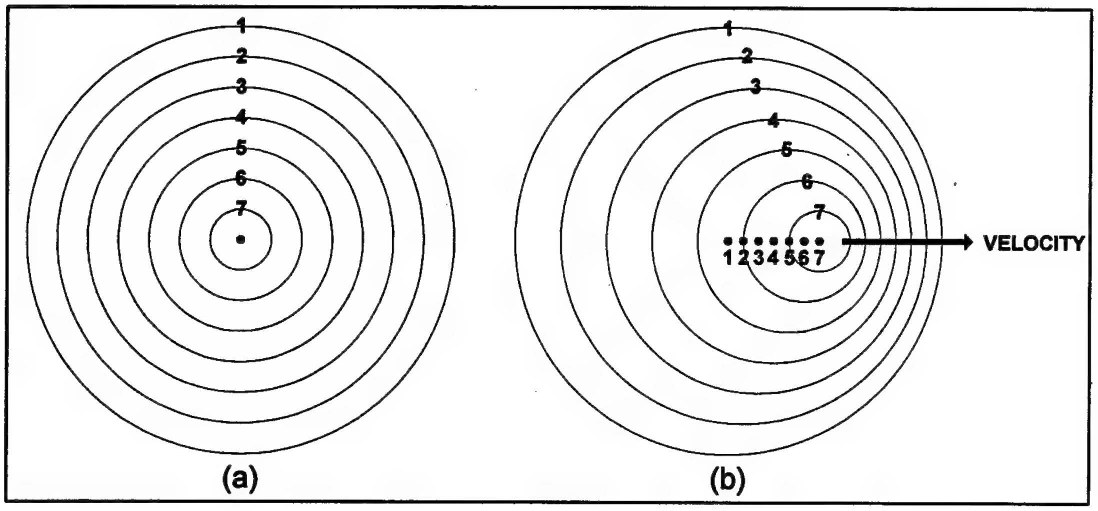


Figure 4 – Waves radiated from stationary (a) and moving (b) point sources. Waves are compressed in the direction of motion.

In the case of a radar, Doppler shifts are produced by the relative motion of the radar and the objects from which the radar's radio waves are reflected. Moreover, a change in the frequency of a wave is equivalent to a continuous shift in phase. For example, if we want to shift the frequency of a wave down from 11 to 10 Hz, we can do so by inserting a time delay equivalent to 36 degrees of phase between successive wavefronts. By shifting phase in the opposite direction – by decreasing the time between successive wavefronts – we increase the wave's frequency.

The same principle applies to a pulse Doppler radar; only in this case, phase is not shifted through the insertion or removal of increments of times between wavefronts, but as the result of the continuous change in time the radio waves take to travel from the radar to the target and back. This time change between pulses is determined by comparing the phase of the received signal with the phase of the reference oscillator of the radar [Skolnik90].

2.6 Space-Time Adaptive Processing - STAP

A radar system sends out a series of pulses and collects the echoes, which return from these pulses on a set of sensors. The returns are sampled, and the resulting data is processed with the objective of determining whether targets are present. The presence of signal interference, which may come from man-made jamming, sensor noise, multipath effects, or the motion of the platform [Ward94] makes this task more complex.

If interference is localized in frequency and comes from a limited number of sources, it can be overcome using *adaptive spatial weighting* of data. The weights applied to the data reduce the effects of interference and increase reception of the desired signal [Haykin91]. For an airborne radar platform, interference due to platform motion is not localized in frequency. In this case, a more thorough approach is to use a sub-array of tapped delay lines connected to each sensor [Haykin91]. The weights may then be adapted from data in both the time and space dimensions. This approach is referred to as *space-time adaptive processing*, or STAP. This technique takes advantage of both the spatial and Doppler diversity of target signal returns, clutter, and interference to extract the desired signal by adaptively combining samples from multiple channels and pulses to null clutter returns and interference. Processing data from multiple channels provides the radar an opportunity to control the spatial response of the system while processing multiple pulses enables the processing to separate signals based upon their Doppler frequency.

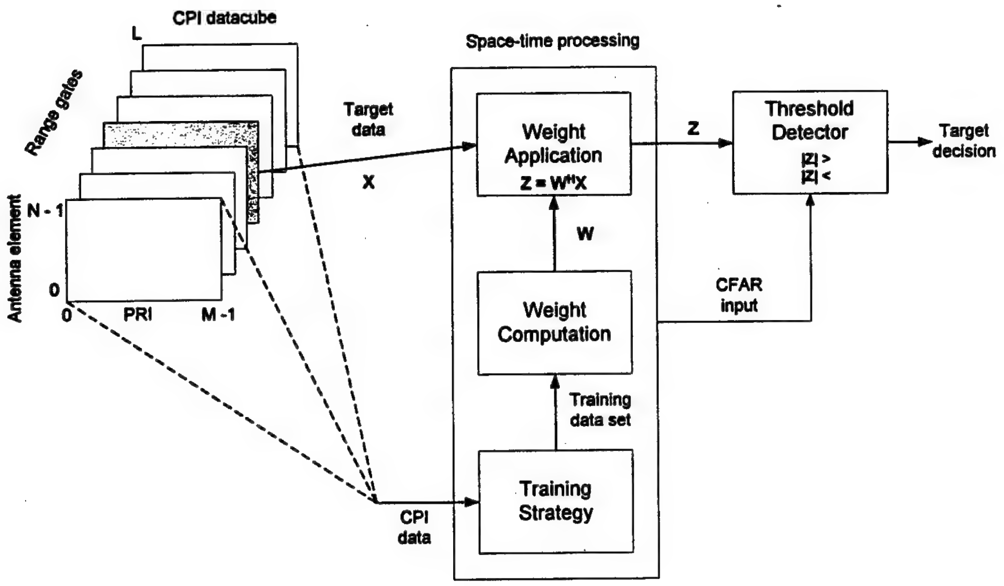


Figure 5 – Space-time adaptive processing overview.

2.6.1 Fully Adaptive STAP

A radar transmits a coherent burst of M pulses at a constant pulse repetition frequency (PRF) $f_r = 1 / T_r$, where T_r is the pulse repetition interval (PRI). The transmitter carrier frequency is $f_0 = c / \lambda_0$, where c is the propagation velocity, and λ_0 is the radar operating wavelength. The time interval over which the waveform returns are collected is commonly referred to as the coherent-processing interval (CPI). The CPI length is equal to MT_r [Ward94].

The space-time adaptive processing is typically performed in one radar coherent processing interval (CPI), which consists of L range gates, M pulse-repetition intervals (PRI), and N antenna elements, as in Figure 5 [McMahon96]. The full CPI data cube is shown in Figure 5 as well as the three major phases of the STAP processing.

More specifically, let x_{nml} be the complex sample from the nth element, mth pulse, at the lth sample time (range gate). Let $\mathbf{x}_{m,l}$ be the N × 1 vector of antenna element outputs, or a spatial snapshot, at the time of the lth range gate and the mth pulse. Now let the N × M matrix $\mathbf{X}_l$ consist of the spatial snapshots for all pulses at the range gate of interest,

$$\mathbf{X}_l = [\mathbf{x}_{0,l} \ , \mathbf{x}_{1,l} \ , \dots , \mathbf{x}_{M-1,\, l}]. \tag{2-1}$$

The shaded slice of the datacube in Figure 5 represents this matrix. The rows of $\mathbf{X}_l$ represent the temporal (pulse-by-pulse) samples for each antenna element. The matrix $\mathbf{X}_l$ is termed a space-time snapshot [Ward94].

Before applying space-time adaptive processing algorithms to data samples measured by the airborne antenna array, a significant amount of *preprocessing* must be performed on the data cube. Typically, it is composed by three major functions applied consecutively in this order: video-to-I/Q (in-phase and quadrature) conversion (comprising demodulation, filtering, and decimation), array calibration, and pulse compression [Cain97].

After the preprocessing phase, a set of rules, called the training strategy is applied to the data. Because the interference is unknown a priori, it must be estimated data-adaptively from the finite amount of data comprising the CPI. The goal of the training strategy is to obtain the best estimate of the interference that exists at the range under test, and usually data from several range gates near the range gate of interest are used.

The second step is weight computation. Based on the training data, the adaptive weight vector is computed through a method called sample matrix inversion (SMI) [Reed74][Ward94:56]. Assuming that at the range gate of interest, a target signal is present in a background of interference, let the target angle, Doppler, and amplitude be given by ν_t, ϖ_t, and α_t, respectively. The data snapshot at the range of interest may be written as

$$\mathbf{X} = \alpha_t \mathbf{v}_t + \mathbf{X}_u \text{, for } \mathbf{v}_t(\nu_t,\varpi_t) = \mathbf{b}(\varpi_t) \otimes \mathbf{a}(\nu_t). \tag{2-2}$$

where the target steering vector $\mathbf{v}_t$ (which is the known response of the system to a unit amplitude target) is equal to the tensor product between the temporal steering vector $\mathbf{b}(\varpi_t) = [1; e^{j2\pi\varpi}; \ldots; e^{j(M-1)2\pi\varpi}]$ and the spatial steering vector $\mathbf{a}(\nu_t) = [1; e^{j2\pi\nu}; \ldots; e^{j(N-1)2\pi\nu}]$, and $\mathbf{X}_u$ denotes the interference plus noise components of the data – clutter, jamming, and thermal noise [Ward94]. If a return from a target is embedded in the returned data, the term $\alpha_t\mathbf{v}_t$ is present on the RHS of the equation 2-2.

The adaptive weight vector **w** for a given steering vector $\mathbf{v}_t$ is related to the interference-plus-noise covariance matrix **R** through the relationship $\mathbf{w} = \mathbf{R}^{-1}\mathbf{v}_t$ [McMahon96]. The covariance matrix **R**, which is unknown a priori, is defined as

$$\mathbf{R} \triangleq \mathrm{E}\{\mathbf{X}_u\mathbf{X}_u^H\}. \tag{2-3}$$

where the operator E{ } denotes the expected value of a random quantity. The use of a covariance matrix is appropriate because target data returned is viewed as a shift in the mean of the data, since from equation 2-2, we note that the target component of the received signal is embedded in interference plus noise components. Thus, in order to best detect the presence of a target, we design a filter which is tuned to the target in such a way that the effects of noise and interference are minimized. One criterion which accomplishes this is the maximum *signal-to-interference-plus-noise-ratio* (SINR) principle. It has been shown [Brennan73] that if the noise and interference approximate Gaussian random processes, than a maximization of the SINR is equivalent to a maximization of the probability of detection. It is in this sense that the maximum SINR criterion is an optimal principle.

The term SMI also refers to more numerically stable algorithms in which the weights are computed from a QR-decomposition [Golub96:223] of the matrix of the training set data – which is the preferred technique for reasons of numerical stability and computational complexity. Typically, weight computation requires the solution of a linear system of equations. This stage is therefore a very computation-intensive portion of space-time processing, as can be exemplified by Table C.2 on **Appendix C**, which includes a discussion on the computational complexity of the different processing stages of the STAP implementation object of this case study.

Finally, given a weight vector, weight application is the formation of the output given the computed weight vectors. The design of the weight application regions is usually coupled with the training set design, with each application region corresponding to a single training data set. Weight application is an inner product, or matrix-vector product, operation involving the weight vector W and the snapshot of interest:

$$\mathbf{Z} = \mathbf{W}^{H}\mathbf{X}\,. \tag{2-4}$$

The computational load of this portion of the space-time processing scales linearly with the weight vector dimension and the number of range gates. The output is a separate scalar for each range, angle, and velocity at which the target presence is to be required. A background noise estimate is provided to the detector so that it provides a constant-false-alarm rate (CFAR).

Therefore, space-time processing can be viewed as a linear combination that sums the spatial samples from the elements of an antenna array and the temporal samples from the multiple pulses of a coherent waveform.

Important features of STAP algorithms are the general architecture, the weight training and application strategy, and the weight computation approach [Ward94].

2.6.2 Partially Adaptive STAP

Although the fundamentals of STAP were first introduced by L.E. Brennan and I.S. Reed in 1973 [Brennan73], STAP has become practical as a real-time technique only with the recent advent of high-performance digital signal processors. Even so, fully adaptive STAP is still beyond the reach of the state-of-the-art processor technology. A radar processor operating in real-time must produce the correct output in a prescribed time limit, or latency. Consequently, sustained performance on key STAP computations and communications patterns is a primary consideration in assessing processor suitability. A STAP signal processor computes a set of adaptive filter weights by solving in real time a system of linear equations of size NM, where N and M are the number of spatial and temporal degrees of freedom, respectively, in the filter. The solution for each set of weights requires on the order of $(NM)^3$ floating-point operations. For the fully adaptive approach a separate set of weights has to be applied to all the antenna elements and all the PRI per CPI. Ward states that the product NM may range from 10^3 to 10^4, leading to a computation requirement of 10^9 to 10^{12} flops with executions speeds of fractions of a second, for the optimal processing algorithm [Ward94]. Figure 6 shows an estimate of the sustained throughput requirements in Gigaflops for next-generation AEW radars [Lockheed98].

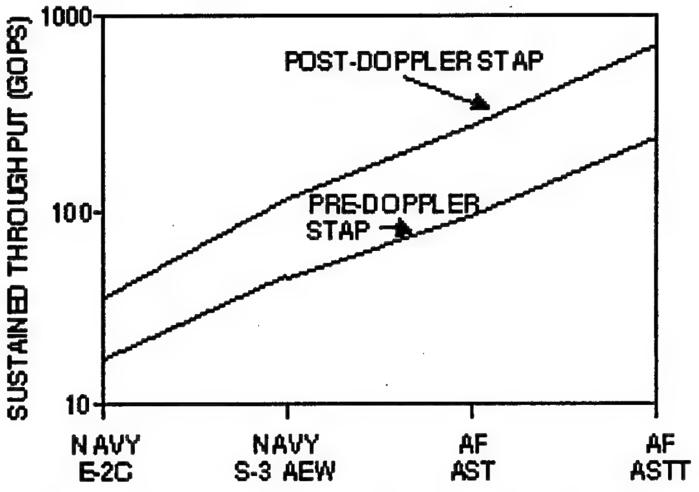


Figure 6 – Throughput requirements for next-generation AEW radars.

Consequently, much of the current research work on STAP has focused on the development of algorithms and associated libraries of routines [Lebak96] that decompose the fully adaptive problem into reduced-dimension adaptive problems by applying heuristics to allow an adequate covariance estimation capable of being implemented in real-time on reasonably sized processors with a limited amount of training data [McMahon96].

There exists four classes of sub-optimal, partially adaptive STAP algorithms, according to the taxonomy provided in [Ward, 1994:90], which are distinguished by the type of suppressive nonadaptive filtering applied after preprocessing and before adaptive processing, or, equivalently, to the domain in which adaptive weight computation occurs.

Each quadrant in Figure 7 shows a box representing the data domain for a single range gate after a different type of nonadaptive transform class. Transitions from one domain to another require the specific discrete Fourier transform (DFT) shown.

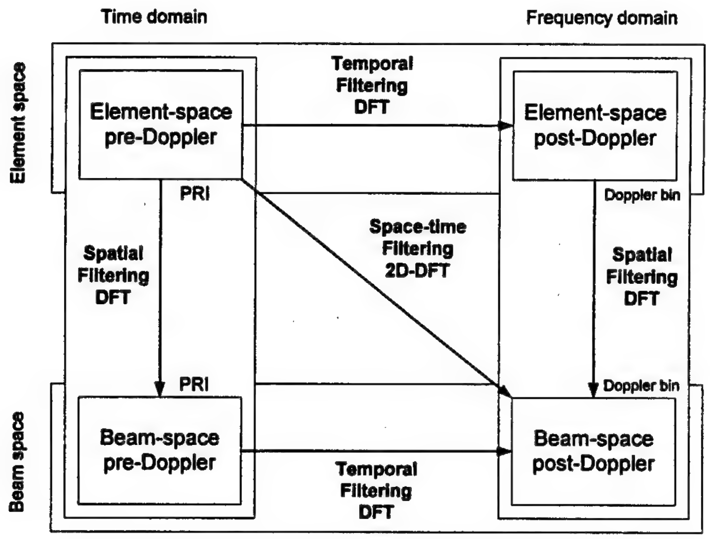


Figure 7 - Taxonomy of reduced-dimension STAP algorithms.

The objective of this nonadaptive transformation, or nonadaptive filtering operation, is to reduce a prohibitively large problem into a number of dimensionality-reduced adaptive problems while achieving near-optimum performance. Many different approaches are possible [Lebak96]. In order to obtain the weight vector the heuristic methods combine elements of the space-time snapshot through a transformation matrix **T** to get a reduced dimension snapshot **X'**,

$$\mathbf{X'} = \mathbf{T}\,\mathbf{X} \tag{2-5}$$

where the matrix **T** may be used only to select certain elements of **X** (e.g., in the case of the upper left quadrant of Figure 7), or it may include a filtering operation [Lebak96] [Ward94:88]. Modern radars can use nonadaptive temporal filtering (Doppler filtering) to isolate the clutter in angle, thus eliminating the interference with the target signal in the mainlobe. As a consequence, less degrees of freedom are needed in the subsequent adaptive processing involving SMI. Figure 8 provides a graphical representation of a generic partially adaptive architecture.

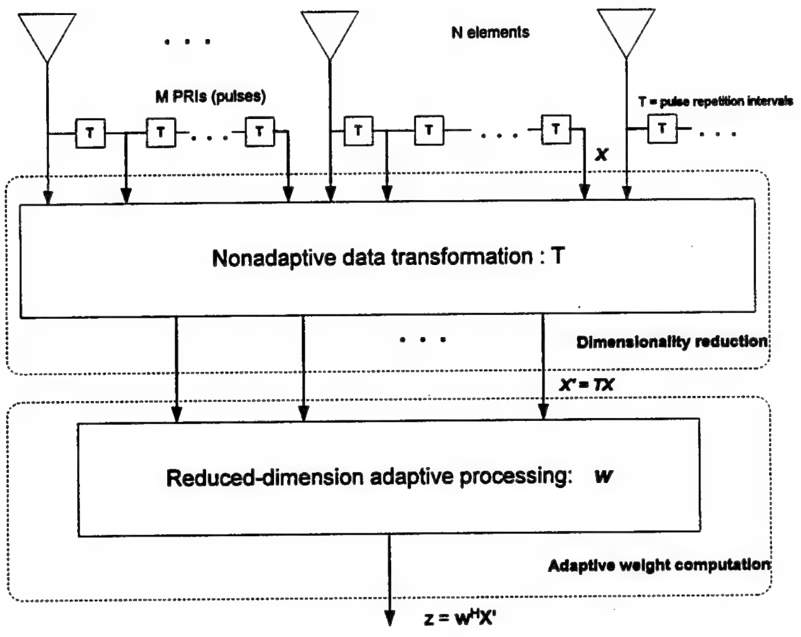

N elements
M PRIs (pulses)
T = pulse repetition intervals
T
X
Nonadaptive data transformation : T
Dimensionality reduction
X' = TX
Reduced-dimension adaptive processing: w
Adaptive weight computation
z = w^H X'

Figure 8 – A generic partially adaptive STAP architecture.

For the partially adaptive STAP, the RMB algorithm [Reed74] is employed, by means of the sample matrix inversion method, where K_e snapshots are used to form the sample covariance matrix estimate **R'** to be inverted. In order to obtain the estimate **R'** to substitute for **R**, we use the maximum likelihood estimate [Papoulis91:260]. The maximum likelihood estimate of **R** has been shown [Goodman63] to be:

$$\mathbf{R}' = \frac{1}{K_e} \sum_{l=1}^{K_e} X_l X_l^H \tag{2-6}$$

From 2-6 we see that the sample covariance matrix **R'** is the arithmetic mean of the K_e sample matrices $X_l X_l^H$. For more details on the specific formation and structure of a covariance matrix the reader is referred to [Papoulis91:188] and [Davenport87:345-348]. Typically, the training samples $\mathbf{X}_l$ cover a range interval surrounding but not including the range gate of interest [Ward94]. This is because, ideally, no target signal should be present in the snapshots used to form the weight vector; that is, only signal interference due to jamming, clutter, and noise should be in the estimation (see equation 2-3). Fortunately, in most radar systems target signals are small and confined to a single range gate [Ward94:77]. On the other hand, current research exists with the objective of enabling STAP to handle a higher density of targets [Stimson98:510]. Because of the covariance estimate, the SMI weight vector is suboptimun.

STAP algorithm functionality can be viewed as a process that provides a gain in SINR, i.e., SINR can be used as a performance metric for STAP. Since **R'** is a function of the random data X_l, SINR is also a random variable. From the probability density function of the normalized SINR, [Reed74] expressed the expected value of SINR as a function of the number of training samples, and concluded that at least 2N samples of data are needed to allow an average loss ratio of less that 3dB, where N is the number of data samples in one space-time snapshot. Thus, the expected loss is independent of any interference scenario, and depends only on the number of snapshots K_e and the weight vector dimension.

The precise training strategy for selecting the snapshot vectors that provides the optimal tradeoff of processing performance and computational complexity is, to some extent, an open question in the STAP community [Cain97].

2.7 Contemporary STAP Research

Adaptive signal processing is an active area of research, comprising efforts in several different fronts, from new algorithm techniques applied to training strategies and weight computation methods to design of specialized hardware capable of meeting the requirements for size, weight, and power (SWAP) for airborne platforms. STAP research is spread among technological institutes, universities, and several corporations.

Parallel computing is deeply associated with STAP, specially because STAP is a computationally demanding task, and requires expertise in many aspects of parallel programming. In general, a efficient STAP parallel implementation is the result of an extensive and carefully conducted design and experimental process, where distributed numerical computing algorithms are at the center, and issues as efficient inter-process communication and load balancing are decisive to good performance.

Other important aspects considered in STAP research include (a) portability and flexibility of the software (considering the different real-world tactical scenarios) [Lebak96][Linder97][Brian98]; (b) software environment for research and development

[CREST96]; (c) general purpose computing platforms vs. specialized DSP hardware for processing [Games98]; (d) real-time processing requirements [Stimson98]; (e) modeling and simulation [Tex98]; (f) use of STAP as a relatively low-cost add-on for performance improvement of conventional antennas [Stimson98]; (g) knowledge-based STAP [Antonik97], and (h) STAP scientific visualization [Napear95].

Additionally, published material exists that is related specifically to lessons learned and results obtained while implementing a given STAP technique in parallel platforms - MPPs, SMPs, and multicomputers - or to the use of different STAP implementations for benchmark purposes [Hwan96][Hwang96]. In this sense, the use of STAP as the case study for this thesis effort can add additional insights to this knowledge base in regard to network-based parallel platforms and portability of software. It is also a step further in the direction of employing high-performance computers instead of specialized hardware in real-time environments. Although difficulties exist, such as operating system overhead and implementation efficiency, the fast development time, flexible nature of software, increased speed and precision, and affordability make the use of these platforms desirable.

The remaining part of this section describes current research and developments concerning STAP, grouped by institution.

<u>Massachusetts Institute of Technology/Lincoln Laboratory (MIT/LL)</u> – The MIT/LL was the first place where a benchmark was developed in C to evaluate Unix workstations for adaptive signal processing [MIT/LL94]. The STAP benchmark consists of three radar signal processing programs: Adaptive Processing Testbed (APT), High-Order Post-Doppler (HO-PD), and General (GEN). These programs start with Doppler processing

(DP), in which a large number of one-dimensional fast Fourier transforms (FFT) computations are performed. The APT performs a Householder transform to generate a triangular learning matrix, which is used in a beamforming step to null the jammers and the clutter; whereas, in the HO-PD program, the two adaptive beamforming steps are combined into one step. The GEN program consists of four component algorithms to perform sorting, FFT, vector multiply, and linear algebra. These are the kernel routines often used in signal processing applications. Table 5 summarizes the workload for this STAP sequential benchmark. Also, research has been conducted on the development of dedicated hardware for STAP with applicability on airborne systems [McMahon96].

MIT/LL also integrates the DARPA/Navy Mountaintop Program, which addresses issues critical to airborne surveillance radar by hosting prototype and development of airborne early warning (AEW) radar and signal processing systems at high-elevation sites. Recently, Northrop Grumman Corp. ordered from Mercury Computer Systems Inc. a 100+ Gigaflops peak RACE© system for research into advanced airborne surveillance radar analysis techniques. This system is going to be used by scientists at MIT Lincoln Laboratories to develop progressive methods for the analysis of STAP algorithms [MNR98].

Program	Component Algorithms	Workload Mflops
APT	Total	1447
	Doppler Processing	84
	Householder Transform	2.88
	Beamforming	1314
	Target Detection	46
HO-PD	Total	12,853
	Doppler Processing	220
	Beamforming	12,618
	Target Detection	14
GEN	Total	5326
	SORT	1183
	FFT	1909
	Vector Multiply	604
	Linear Algebra	1630

Table 5 – Workload for the sequential STAP benchmark from MIT [Hwang96].

University of Southern California (USC) – Parallel versions of the sequential STAP benchmark from MIT were developed, and have been used to evaluate parallel architectures (specifically MPPs), programming environments, and message passing libraries for signal processing applications [Hwan96][Hwang96].

Rome Laboratories, NY – Research is being conducted towards the development of efficient parallel implementations for STAP as well as portable parallel libraries for adaptive signal processing [Linder97][Lebak96]. Currently, a parallel pipelined version is available, implementing PRI-staggered post-Doppler STAP based upon input datacubes provided by the Multi-Channel Airborne Radar Measurements (MCARM) data collection systems. Performance results using up to 236 nodes of the Rome Labs' Intel Paragon MPP are available in [Chouldhary97]. The Air Force Research Laboratory (AFRL) in Rome, NY, has plans to perform STAP research using a recently acquired Sky© parallel computer platform (considered the Air Force's fastest computer and approximately the number 20 in the world). The parallel machine comprises 384 333 MHz IBM PowerPC© 604e processors and 50GB RAID storage system, being capable of deliver 256 Gflops in its initial configuration (www.sky.com/news/Rome.html).

MITRE Corporation, Bedford, MA. – Work has been done in order to develop benchmark methodologies and specifications for real-time embedded scalable high performance computing, STAP benchmarks, and common operating environments for real-time signal and image processing [Games96][Cain97][Games98]. Specifically, the work in [Cain97] provides a specification of the real-time STAP benchmark, RT-STAP.

The benchmark comprises three different STAP implementations, therefore involving three levels of complexity: easy, medium, and hard. The easy benchmark corresponds to the post-Doppler adaptive Displaced Phased Center Antenna (DCPA) algorithm and requires a real-time computational throughput of 0.60 Gflops/sec. This case represents technology used in current radar systems. The medium benchmark case corresponds to the first-order Doppler-factored STAP and requires a throughput of 6.46 Gflops/sec. The hard benchmark case corresponds to an implementation of the third-order Doppler-factored STAP and requires a throughput of 39.81 Gflops/sec. The RT-STAP also includes the implementation of the data preprocessing typically performed before the application of nonadaptive filtering and subsequent adaptive processing. Table 6 summarizes other quantitative information about this benchmark.

For the first-order Doppler-factored STAP and third-order Doppler-factored STAP parallel implementations are available that support up to 64 processors and include software interfaces implementing function name resolution [Brian98] to allow the use of specialized linear algebra routines designed for the Sun©, Mercury©, and Sky© computing platform vendors.

Function	Workload (Gflops/sec)		
	DPCA	First-Order Factored STAP	Third-Order Factored STAP
Video to I/Q conversion	0.22	1.77	2.43
Calibration and Pulse Compression	0.26	2.10	2.88
Preprocessing Total	**0.49**	**3.87**	**5.31**
Doppler processing	0.06	0.49	0.67
Weights Computation	0.03	1.98	33.33
Weights Application	0.02	0.12	0.50
STAP Total	**0.11**	**2.59**	**34.50**
Total	**0.60**	**6.46**	**39.81**

Table 6 – Workloads for the three RT-STAP benchmark cases [Cain97].

Texas Tech University, TX. – Research sponsored by DARPA exists towards the investigation of the advantages of integrating configurable hardware – FPGA-based boards – with multiprocessor GPP/DSP-based platform (General Purpose Processor/Digital Signal Processor). Current research directions include: optimal configuration of parallel embedded systems for Synthetic Aperture Radar – SAR – processing; simulation of communication time for STAP onto a parallel embedded system; implementation and evaluation of a power prediction model for a FPGA;

reconfigurable computing for STAP, and novel computing techniques based on characteristics of future GPP/DSP/FPGA systems [Tex98].

Vector Signal Image Processing Library Forum (VSIPL) – The VSIPL is a volunteer organization made up of industry, government, users, and academia representatives who are working to define an industry standard API for vector, signal, and image processing primitives for embedded real-time signal processing systems. Some of the goals include the creation of a widely (industry) supported standard API/library for single processors and parallel versions, therefore fostering the standardization, reuse, interoperability, low cost COTS upgrade paths, and lower life cycle costs for related applications [VSIPL98]. In that sense, versions of the RT_STAP benchmark from MITRE Corp. were implemented using the VSIP reference library, as reported in [Kenneth97].

IEEE National Radar Conference – Approximately one third of the material published on the proceedings for this conference, sponsored by the Aerospace and Electronics Systems Society, is directly related to STAP, produced by government research institutions and contractors, as well as different corporations like Lockheed Martin, Northrop-Grumman, Kaman, and MITRE. Contents of the 1997 and 1998 conferences include papers on algorithm development and performance evaluation for STAP, alternative methods to SMI for covariance estimation and weight computation, STAP performance in nonhomogeneous interference environments, statistical modeling of ground and sea clutter interference, knowledge-based STAP, alternative mathematical approaches to STAP, and interference rank estimation. The proceedings of the 1998 conference also include papers describing research done by AFIT, and refer to the use of the signal-to-interference plus noise ratio (SINR) metric as a tool for the selection of rank reduction transformations [Scott98], as well as secondary data support in STAP [Welsh98].

From this survey we can see that research on space-time adaptive processing is much more concentrated on the theoretical or experimental aspects of the problem, although there exists work related specifically to computational aspects. As we already know, *heuristics* is an important component of adaptive signal processing algorithms, and the performance characterization of heuristic algorithms is a complex issue that tends to cluster in the areas of solution quality, robustness, and *computational effort* [Barr95]. Moreover, computational aspects gain additional importance in the STAP problem because it is a real-time application that encompasses several stages (each one with different computational needs and a possible bottleneck). Therefore, our belief is that STAP research should benefit from research done by people from both the real-time and the high-performance computing communities working together.

Concerning parallel architectures, **Appendix A** presents an overview of the existing landscape in terms of parallel computer architectures in order to provide a basic understanding of the different directions of the architectural evolution, the forces that determine the path for design decisions, and the impact of these decisions on performance-oriented programming. In particular, fundamental issues like locality of reference, bandwidth, latency, and synchronization are mentioned with the objective of being useful to a more complete understanding of this thesis work. **Appendix B** provides a detailed description of the ABC, AFIT NOW, and IBM SP parallel computational platforms used in this research.

2.8 Summary

This chapter provided a literature review on STAP and basic radar operation. In particular, the motivation for radar adaptive processing, its general computational complexity, different algorithmic approaches, and taxonomy were discussed. Digital signal processors and parallel signal processing were considered in terms of the their power as resources to problem solving. Finally, an overview of parallel architectures and its different inherent characteristics was presented through reference to Appendix A. The next chapter – Methodology - reports the overall rationale behind this thesis research, and a detailed description of the selected STAP implementation to be employed as the case study, according to its specific objectives.

III. Methodology

3.1 Introduction

The objective of this chapter is to report the methodology that guided this thesis investigation, and it is organized in two sections. The first part details the actions taken in order to achieve the objectives of the research. Therefore it contains a description of the aims, order, and nature of the work performed . Different alternatives for research are considered, and decisions made are justified. The second part deals with the problem domain / algorithm domain integration, and contains a detailed description of the selected STAP implementations, specifications, and associated complexities.

3.2 General Methodology

As already mentioned in the first chapter, to investigate the use of the ABC for a spectrum of signal processing applications, using STAP as the case study, involves three basic steps: obtain a representative implementation of STAP, port it to the cluster, and check for effectiveness and performance figures. The last step can be explored further by modifying the source code of the original implementation in order to allow it to get improved performance from the platform to which it was ported while maintaining the original portability. Thus, the discussion in this section is organized around these steps.

3.2.1 Selection of the STAP Implementation

The STAP implementation selected was provided via the AFRL Rome Laboratories, NY, and is the Real-Time Space-Time Adaptive Processing Benchmark – RT_STAP – created at MITRE Corporation and currently in version 1.1. It is a realistic compact application benchmark based upon data collected by the Rome Laboratory

MultiChannel Radar Measurement Program (MCARM) airborne system. This system was developed by the Westinghouse Electronic Systems Group (now Northrop-Grumman, Electronic Sensors and Systems Division, Maryland) for the purpose of collecting and recording L-band (1.3 GHz) radar returns transmitted from an airborne platform. The data measured by the MCARM data collection system can be used to evaluate the ability of STAP techniques to cancel Doppler-spread clutter and interference [Cain97]. Rome Laboratory has collected a number of useful data sets and has made them available for processing. A selected data set was chosen to provide the input for RT_STAP implementations. The RT_STAP v. 1.1 is written in C, and provides sequential versions of algorithms implementing DPCA processing, first-order post-Doppler, and high-order post-Doppler factored STAP, as well as parallel versions of the last two algorithms mentioned, using MPI as the communication mechanism.

The primary reason for this selection is the fact that this application was developed specifically to evaluate the performance of high-performance computers for STAP [Cain97], including key features such as (a) the incorporation of preprocessing of the data cube before adaptive processing; (b) three different adaptive processing implementations corresponding to increasing computational complexity and workload levels; (c) the specification of real-time requirements in accordance to the parameters of the MCARM data collection system. Our belief is that features (a) and (c) provide the necessary *representativeness*, and feature (b) the *potential*, for a number of experiments in this case study.

Other alternatives in terms of STAP implementations were the parallel pipelined STAP from AFRL Rome Laboratories, and the sequential MIT STAP benchmark. The first requires seven processors for minimal execution configuration (the number of pipeline steps of the algorithm), which would impair the investigations based on the machine scalability, given the overhead incurred by the use of PCs with slower I/O buses (66 MHz against 100 MHz). The second has less constraining real-time computational requirements when compared to RT_STAP, presents just one type of adaptive algorithm (high-order post-Doppler), and also would require considerable time for the parallelization of several component kernels.

3.2.2 Porting to the ABC

The second basic step in our methodology is to port, or install, the RT_STAP benchmark to the ABC. A decision was made to adopt the Linux 2.0.33 mode of the cluster. Since this operating system is a Unix version for the Intel architecture, the time spent in this process should be minimum and focused on resolving any possible compiler differences in regard to the code and compilation flags, and the edition of the *Makefile*, for it to point to the appropriate MPICH 1.1 communication libraries and base directories. The adoption of the Windows NT mode and corresponding MPI implementations (MPI Pro or PaTENT) was discarded due to the larger overhead associated with this operating system, as showed by an investigation performed by the DOE Ames Laboratory using the protocol-independent, network performance analysis tool named NetPIPE, and available at http://www.scl.ameslab.gov/Projects/ClusterCookbook/icperf.html.

The building process should be directed to use the single-precision standard ANSI C implementation of STAP in order to reflect the default validation criteria of the benchmark [Cain97:54], and to cope with the fact that the ABC does not have a set of customized library routines to perform linear algebra and signal processing operations. At a later stage of this work, however, the software may have to be rebuilt in order to be able to use any library functionality made available to the cluster and used through modifications on the original source code.

3.2.3 Effectiveness

The next step is to run the different implementations contained in the software package on the ABC, and check for effectiveness and performance of the results obtained for both sequential and parallel versions. The aspects concerning performance are discussed in Chapter IV – along with the respective metrics - and here we concentrate on effectiveness, that is, correctness of the results.

The RT_STAP implementation supports self-validation as one of the option flags (-*v*) on the command line for execution [Cain97], and this feature was used throughout this thesis effort to test for correctness of the results generated by the benchmarks.

The validation process compares the range-Doppler results generated by program execution with pre-computed double-precision results generated using the sequential software and stored in files. Let $z(k,r)$ be the processed radar return associated with Doppler bin k and range cell r generated using the double-precision version of the sequential software. Let $\hat{z}(k,r)$ be, for example, the radar return computed using a single-precision parallel software implementation. The validation criterion requires that the peak change in power values (measured in dB) be less than 1 dB for power values greater than a specified threshold (i.e., 10 dB below the estimated noise floor P_N). Specifically, the validation criterion is given by:

$$\max_{k,r} \left| 10\log_{10}|z(k,r)|^2 - 10\log_{10}|\hat{z}(k,r)|^2 \right| < 1,$$

where the maximum is over all values of k and r such that

$$10\log_{10}|z(k,r)|^2 > (P_N - 10) \text{ or } 10\log_{10}|\hat{z}(k,r)|^2 > (P_N - 10).$$

The noise floor is estimated over a region in the range-Doppler map that is removed from the mainlobe clutter. For this benchmark, P_N is estimated as

$$P_N = 10\log_{10}\left(\frac{1}{\lfloor 0.2N_D \rfloor \cdot \lfloor 0.2K \rfloor} \sum_{k=\lceil 0.3K \rceil}^{0.5K-1} \sum_{r=\lceil 0.8N_D \rceil}^{N_D-1} |z(k,r)|^2\right).$$

where N_D is the number of samples per pulse after decimation, and K is the Doppler filtering FFT size. The post-Doppler adaptive processing range-Doppler results $z(k,r)$, calculated using the double-precision version of the sequential software, are provided with the package.

3.2.4 Modifying the STAP Code

Signal processing computer applications are characterized by heavy use of mathematical operations, notably numerical linear algebra calculations involving vectors and matrices, as well as others related to the field like filtering and linear transformations. Since these computations often involve repetitive operations on possibly large data sets, inefficient implementation of these routines can significantly degrade the performance of the programs that use them.

One of the important aspects to utilizing a high-performance computer effectively is to avoid unnecessary memory references [Dongarra98]. The movement of data between memory and registers can be as costly as arithmetic operations on the data. This cost provides considerable motivation to restructure existing algorithms in order to better explore spatial locality within the memory hierarchy. In this sense, the STAP code is no exception to the rule.

Close examination of the source code reveals the existence of several vector-scalar, vector-vector, and matrix-vector linear algebra operations, and also signal processing tasks where the fast Fourier transform (FFT) [Cooley65] is a frequently used kernel. Since one of the objectives is to modify the original code to allow it to obtain better performance while running on the ABC, the aforementioned routines are the targets for alterations in order to produce an implementation that better utilize the resources of the host processor and of the memory hierarchy (especially caches). Therefore, the idea is to provide the ABC with a Basic Linear Algebra Subroutines (BLAS) Library and a set of customized FFT routines that could explore the capabilities of the Pentium II© processor, and modify our application to make use of it, where applicable.

The adoption of this practice also offer several other benefits. *Robustness* of linear algebra computations is enhanced by the use of specialized libraries, since they take into consideration algorithmic and implementation subtleties that are likely to be ignored in a typical application programming environment, such as treating overflow situations, and program *portability* is improved through standardization of computational kernels without giving up efficiency, since optimized versions of these routines can be used on those computers for which they exist. Finally, program *readability* and *modularity* are improved, because these libraries are development tools; that is, they are a conceptual aid in coding, allowing one to visualize mathematical operations rather than the particular detailed coding required to implement it. Often, widely recognized mnemonic names are associated with the operations, which also improve the self-documenting quality of the programs.

The rationale behind the decision of applying modifications to the source code is twofold. First, it is a way to add a new important factor to the study, other than only conduct a workload-driven evaluation for sequential absolute performance and performance due to parallelism. Other types of modifications that could have been considered are those related to the data structures used, data layout distribution, and communication/synchronization [Culler98:219]. Second, workloads often vary substantially across the range of performance-related characteristics (e.g. platform computing node performance, inter-process communication, synchronization, etc.). Modifications in the source code can amend the unbalance that may exist when one chooses a workload that, although representative of a domain, stresses features for which the host architecture has an advantage / disadvantage [Culler98:219]. Additionally, this procedure may create the need for new or different performance metrics.

It is also important to state that the libraries downloaded and installed on the ABC for the purpose of this study are not necessarily the ones that deliver optimum performance, for this should be object of a separate investigation that is beyond the scope of this thesis work. Nevertheless, we analyze the effectiveness/efficiency of these packages in regard to aspects that are critical to the objectives of the proposed research, as detailed on the following two chapters. The next two subsections provide information on the BLAS and FFT libraries installed on the ABC and used to build the modified version of RT_STAP.

3.2.4.1 The BLAS

One way of achieving efficiency in the solution of linear algebra problems is through the use of the Basic Linear Algebra Subprograms. In 1979, Lawson et. al.

[Lawson79] described the advantages of adopting a set of basic routines for linear algebra problems, as we addressed in the last subsection.

For the purpose of this thesis effort, we download and install on ABC a BLAS implementation from the Sandia National Laboratories, in Albuquerque, New Mexico (http://www.cs.utk.edu/~ghenry/distrib/index.htm). This FORTRAN implementation, in its version 1.1L from November 1998 and called ASCI Red Pentium Pro© BLAS, is one of the utilities and libraries developed for the Intel ASCI Option Red Supercomputer (comprised of 9216 Pentium Pro© processing nodes), and is targeted to Unix-like environments. This library is also used by the California Institute of Technology's Beowulf cluster named Naegling (http://www.cacr.caltech.edu/beowulf/naegling.html).

Once installed, this library was tested to validate its basic functionality and the effectiveness of the results generated during computations. Also, a comparative analysis (focusing performance) against a reference FORTRAN implementation from the Netlib (http://www.netlib.org) software repository was accomplished, and the results are reported on Chapter V. For these tests, two versions of the LAPACK package from Netlib were built on ABC: one using the reference implementation and another using the ASCI Red Pentium Pro© BLAS. The LAPACK package comes with an extensive set of test (for correctness) and timing (for performance) programs that are useful for BLAS testing and timing purposes as well [Dongarra94:15]. LAPACK uses the BLAS as the building block for its routines, exploring all three levels of BLAS, using the highest level at all times [Higham96].

3.2.4.2 The FFT

With the primary objective of improving the performance of the FFTs presented in the RT_STAP code, we downloaded the necessary routines and respective dependencies from a publicly available FFT package from Netlib (http://www.netlib.org/fftpack), named FFTPACK. It is a reliable and widely used [Briggs95:397] set of FORTRAN implementations based upon a radix-2 version of this algorithm, which is by far the most widely used [Proakis96:456], but not as efficient as the mixed-radix version [Lamont97]. The algorithms in this package were created to work with arbitrarily sized arrays of points, i.e. they work also for sizes that are not powers of two, and were originally developed for elliptic partial differential equations, containing also sine, cosine, and quarter-wave transforms [Swarz82].

After being downloaded, the routines were compiled using *f77* with optimizations (the flags were *–O3 –malign-double –fomit-frame-pointer*) and archived into a library (*.a* file) to be available to the modified RT_STAP programs.

Likewise with the BLAS, we performed timing tests in order to address the performance of these routines - specifically, forward and inverse single precision complex 1D-FFTs – by comparing the Mflops rate achieved while operating on arrays of various sizes. For the purpose of comparison, we executed the same test above using the corresponding C-interface FFT routines provided by the Intel Math Kernel Library© (MKL) package for Windows NT operating system. In fact, many of the routines present in the ASCI Red Pentium Pro© BLAS, mentioned in the last subsection, are Unix conversions of BLAS routines contained in this package from Intel, according to information in the Sandia Laboratories download site at the address http://www.cs.utk.edu/~ghenry/distrib/archive.htm.

	FFTPACK	MKL
Source	Netlib http://www.netlib.org	Intel http://developer.intel.com
Language	FORTRAN 77	C
Compiler	GNU f77	MSVC 6.0
Compiler Optimizations	*–O3 –malign-double –fomit-frame-pointer*	/W3 /Gx /Ob2 /D WIN32 /D plus inline function expansion
Operating System	Linux 2.0.33	Windows NT 4.0

Table 7 – Parameters for the FFT used in comparisons.

By this comparison we intended to check whether the results of the compiler optimizations applied on FFTPACK can provide reasonable efficiency when compared to the MKL performance (assumed to be "optimal" for the Pentium II© processor), even though the operating systems and the programming language used are different, as described in the Table 7. The MKL provides FFT routines written in FORTRAN as well, but they could not be used under the Windows NT because a FORTRAN compiler (like the Intel's *ifl*) was not available for that environment.

Before discussing particular aspects related to the RT_STAP specifications, such as the scaling of workload levels and timing, the reader should refer to **Appendix C**, where a detailed description of the STAP algorithms and associated complexities is provided, together with the dynamic of the communication and I/O operations.

3.3 Benchmark Specifications: Timing and Workload

The parameters used to define both the timing specification and the workload study are based upon the MCARM data collection system [Cain97]. The system is comprised of a 32 element antenna, transmitters, and receivers capable of generating L-band (1.3 GHz) radar measurements. The 32 antenna outputs are combined to form 22 channels. MCARM transmits a linear FM signal with a pulse repetition rate ranging between 250 and 2000 KHz and pulsewidth of 50 or 100 microseconds. The system also contains a recording system that enables the operator to record up to a 100 millisecond CPI data block from all 22 channels, at a rate of 5 MHz. The data recording system can store a single CPI every 2 seconds. The system is mounted on a BAC 1-11 aircraft and can be used to collect airborne radar data suitable for evaluating the performance of STAP algorithms. A number of data collection experiments have been conducted in the Chesapeake Bay area and several data sets have been made available by AFRL Rome Laboratory.

Table 8 summarizes the parameter values used to determine timing requirements and evaluate the throughput for both the preprocessor and STAP algorithm when applied to the MCARM example. Note that the MCARM array provides IF data consisting of real A/D samples generated with a sampling rate of 5 MHz. After conversion to baseband, the I/Q is decimated by a factor of four to reduce the sample rate to 1.25 MHz.

Parameter	Value
Number of channels to be processed (L)	see Note 1
Number of pulses per Doppler processing block (P)	64
Number of channels in binary file containing input data cube	22
Number of PRIs in binary file containing input data cube	65
Number of samples per pulse before decimation (N)	1920
Decimation factor (D)	4
Number of samples per pulse after decimation (N_D)	480
FIR filter length used in video-to-I/Q conversion (K_a)	36
FIR filter length used in array calibration (K_c)	3
FIR filter length used in pulse compression (K_p)	63
Convolution length used to implement calibration and pulse compression ($\tilde{R}$)	192
FFT size used by the overlap-save method (R)	256
Number of blocks used to implement the overlap-save method (B)	3
Doppler FFT size (K)	64
Number of independent non-overlapping range blocks (M)	see Note 2
Number of range cells per weight computation (N_R)	see Note 3
Processing Order (Q)	see Note 4

Note 1: L for the hard, medium, and easy benchmark cases is 22, 16, and 2, respectively.

Note 2: M for the hard, medium, and easy benchmark cases is 2, 6, and 6, respectively.

Note 3: N_R for the hard, medium, and easy benchmark cases is 240, 80, and 80, respectively.

Note 4: Q for the hard, medium, and easy benchmark cases is 3, 1, and 1, respectively.

Table 8 – Parameters for RT_STAP.

For the evaluation of the hard, medium, and easy benchmark cases, 22, 16, and 2 of the 22 available MCARM data collection channels were used, respectively. For all three cases, the CPI consisted of 64 contiguous pulses. The high performance computer must input 0.49, 3.93, and 5.41 Mbytes of real data per CPI for the easy, medium, and hard benchmark cases, respectively.

Another aspect to consider for this real-time benchmark is the specification of the period and latency required by the application. The period is defined to be the time between input data sets while latency is the time required to process a single data set [Cain97]. The latency corresponds to the time from when the first data leaves the data source to the time the final result is output to the data sink. The strictness of the latency requirement determines the difficulty and the feasibility of the parallel implementation.

For RT_STAP both the period and latency are closely associated with the CPI of the radar system. The period corresponds to a single CPI, and according to the MCARM specifications, it equals 32.25 milliseconds corresponding to a CPI with 64 pulses. The latency case requirements dictates that data input, processing and writing to data sink must occur within 5 CPIs. This corresponds to a latency of $32.25 \times 5 = 161.25$ milliseconds for the MCARM.

The perspective in terms of workloads for RT_STAP is devised with the objective of explicitly treat real-time requirements, and the approach used towards scalability is problem-centric, that is: for a given problem, the objective is to determine the machine size needed to meet a specified timing requirement, and then scale the problem complexity (including the real-time requirements). A more thorough discussion on this matter is provided in the next chapter, when performance metrics are presented.

The operation rates are specified in billions of floating-point operations per second (Gflop/s) and are computed by dividing the operation counts by the period. For this scenario, the period is equivalent to the duration of the CPI and is 32.25 milliseconds.

As an example, let us calculate the operation rate for weight computation using the high-order post-Doppler STAP. Using the formula for the operation counts for this phase, $K \cdot M \cdot \left(8 \cdot [L \cdot Q]^2 \cdot (N_R + 1)\right)$, and applying the corresponding parameters of Table 8 we get a result of 1,074,991,104 floating point operations. Dividing this value by the period, 32.25 milliseconds, we obtain the required operation rate of 33.3333 Gflops/sec.

For the easy, medium, and hard benchmark cases, the preprocessing and the post-Doppler STAP algorithms require a total operation rate of 0.60, 6.46, and 39.81 Gflop/s, respectively. For the easy case, preprocessing dominates the computation complexity, accounting for over 82 percent of the processing operation rate. For the medium benchmark case, first-order Doppler-factored STAP accounts for slightly less than 40 percent of the processing required for the 16 channel data cube (i.e., 2.59 Gflop/s for the STAP algorithm versus 3.87 Gflop/s for the preprocessing). For the hard benchmark case, the computational complexity of the STAP algorithm significantly increases and accounts for nearly 87 percent of the processing required for the 22 channel data cube (i.e., 34.50 Gflop/s for the STAP algorithm versus 5.31 Gflop/s for the preprocessing).

Benchmark	# of QRs	Matrix Size
DPCA processing	384	80 x 2
First-order STAP	384	80 x 16
High-order STAP	128	240 x 66

Table 9 – Number and size of QR-decompositions for the three benchmark cases.

These values are summarized in Table 6 from Chapter II. Table 9 lists the number and size of the QR-decompositions required to compute the adaptive weights for each benchmark case.

3.4 Summary

In this chapter we presented our three-step methodology for evaluating the ABC using STAP as the case study. First, the criteria used for application selection and the porting of the RT_STAP implementation were presented. Different alternatives for these processes were also discussed. Second, the RT_STAP complexity analysis, and the BLAS and FFTPACK libraries were considered. Finally, the specific RT_STAP benchmark requirements were defined as a function of the algorithm selected and the size of the input. The next chapter – Experimental Framework – discusses the metrics and the experimental design to which the RT_STAP has to adhere in the course of the performance analysis.

IV. Experimental Framework

4.1 Introduction

The objective of this chapter is to establish the knowledge base necessary to structure appropriate experiments, and statistically evaluate the results. Therefore, it contains the definition of the performance metrics and the design of the experiments required to provide some quantifiable measure from which to perform statistical analysis and draw conclusions, according to the level of observation and the fundamental hypotheses of the research. The chapter concludes with a section discussing assumptions.

4.2 Design of the Experiments

An experimental design consists of specifying the number of experiments, the factor level combinations for each experiment, and the number of replications of each experiment [Jain91:277]. The objective is to obtain the maximum information with the minimum number of experiments - according to the confidence levels and accuracy desired, allow reproducibility, and generate supportable conclusions.

To ensure that the reported information is meaningful, we should document and use an experimental design that considers as many factors as is possible and practicable for the time available [Barr95]. We initiate our discussion by considering the factors and levels for the experiments with the RT_STAP implementation. In general, the response variable for our experiments is the *execution time* of whole programs, or timings from specified sections of them, for we are comparing the performance of different algorithms for the same class of problems. Therefore, we have chosen factors that affect the response variable in accordance to the level of observation for the research, and they are: (a) the different *implementations* of adaptive processing contained in the RT_STAP – first-order post-Doppler, high-order post-Doppler factored STAP, and DPCA processing; (b) the machine configuration – specifically, the *machine size* in number of processors applied to the problem; (c) the different *versions* of the implementations – the original and the modified versions of the programs.

The factor *implementation* and its three levels is probably the most important one, because it incorporates scaling in the size of data input for the problem as well as increased computational complexity. Usually, the analysis of parallel programs employ the factor *machine size*, for it enables one to make inferences on machine scalability and improvements in execution times due to parallelism (speedup). The factor *version* is included because we want to quantify the effects of its action on the performance of the programs (the significance), as well as the interactions that it may have with the other factors.

After the factors and levels have been chosen, the next step is to decide which kind of experimental design to use. This effort uses the *full factorial* design, where every possible combination of the factors is examined. A graphical representation of this organization is provided in Figure 9 for the RT_STAP.

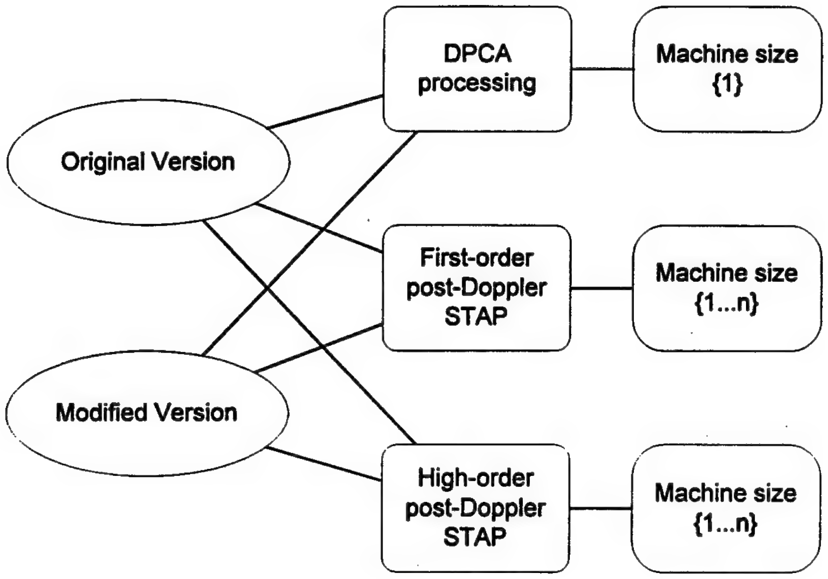


Figure 9 – Full factorial design of the RT_STAP experiments on ABC.

Strictly speaking, the full factorial design for this experiment is not realized, because the DPCA processing implementation is not provided in a parallel version, and instead of 42 experiments (2 versions × 3 implementations × 7 machine sizes), we may accomplish up to 30. For the purpose of comparisons and additional insights, only the *original* versions of the STAP programs are to be ported and executed on the alternative platforms (the AFIT NOW and the IBM SP) though, since the primary objective is to observe the behavior of the STAP programs in regard to different communication and computation performances of those machines.

4.2.1 Statistical Analysis

The data analysis and interpretation steps of the research are the culmination of all the planning and implementation activities and, in the end, determine the overall merit of the work [Barr95]. For this reason, it is important to use statistical analysis as a tool that can help us provide a justifiable rationale and logical explanations for the results observed during the experimental phase.

The *mean* is the measure of central tendency most largely used in this work to average data collected from experiments. Therefore, it is important that it always be informed along with measures of variability, like the standard deviation. The variability of the execution times are of great interest for real-time applications.

The standard deviation is also useful to allow the calculation of the confidence intervals for the means, an important tool for statistical inference. For the purpose of this work, a 95% confidence interval for the means is calculated and informed where applicable.

Means and standard deviations are single summaries, for they focus on just one aspect of the data. An important tool for viewing a distribution of the samples is a *boxplot* [Devore95:33], because it can describe several of the data set's most prominent features: (a) center; (b) spread; (c) the extent and nature of any unusually departure from symmetry; (d) identification of outliers that can affect the validity of some statistics – particularly the mean.

Boxplots are built around measures that are resistant to few outliers – the *median* and the *fourth spread*. For the purpose of this thesis research, boxplots are used to check for the presence of possible outliers that, according to the nature of our experiments, should be lightly positively skewed and rare because the ABC is a dedicated platform.

Another aspect to be considered is the validation of the assumption of normal distribution of the samples collected in the experiments. This validation is important because many of the statistical procedures conducted in this work are based on the assumption of normality. Additionally, understanding the underlying distribution can sometimes give insight into the physical mechanisms involved in generating the data. An effective way to check a distributional assumption is to construct what is called a probability plot. Formally, a *normal probability plot* is a plot of the *n* pairs

$$\left(\left[100(i-0.5)/n\right]^{th}\ z\ percentile\ ,\ corresponding\ observation\right)$$

on a two-dimensional coordinate system. If the sample observations are in fact drawn from a normal distribution with mean value μ and standard deviation σ , the points should fall close to a straight line with slope σ and intercept μ [Devore95:186].

How much the point (in the probability plot) can deviate from a straight-line pattern is not an easy question to answer, though. The work in [Cuthbert80] presents the results of a simulation study in which numerous samples of different sizes were selected from normal distributions. The authors concluded that there is typically greater variation in the appearance of the probability plot for sample sizes smaller than 30, and only for much larger sample sizes does a linear pattern generally predominate. Given the desired requirements for the means, we determine the sample size *n* for an experiment by using the formula

$$n = \left(\frac{100 \cdot z \cdot s}{r \cdot \mu}\right)^2 \qquad (4-1)$$

where z is the normal quantile for 95% confidence level (1.960), s is the standard deviation of the samples collected, r is the desired accuracy (0.001 for this study), and μ is the mean of the samples collected [Jain91:216].

Finally, the Analysis of Variance (ANOVA) is also applied in this study. It refers to a collection of experimental situations and statistical procedures for the analysis of quantitative responses from experiments, and are used for: (a) testing for the hypothesis of equal means between samples with different treatments; (b) validating observations based on the F-distribution (F-test); (c) characterizing the variability of the results caused by different factors in isolation, by interaction, or random fluctuation. In this sense, two-way analysis is performed. Thus, this thesis research considers both first-order and second-order statistics on data analysis.

4.2.2 Software Support

The use of data analysis tools can speedup the work and improve the accuracy of the statistical analysis phase. Most of the data from experiments are stored and manipulated in spreadsheets using Microsoft Excel©, which was also set up to perform some specific tasks like creating boxplots, normal probability plots, and performing ANOVA calculations.

Visualization of parallel program execution is another important aspect related to experimental design. While simulation and analytical models are routinely validated before use, validation of measured data is rarely thought of [Jain91]. In this sense, this thesis effort applies the Vampir© tracing and visualization software to provide persuasive insight in regard to program execution, inter-processor communication and synchronization, and cross-checking and validation of timing measurements.

4.3 Performance Metrics

For a parallel machine, we can measure two performance characteristics: the *absolute performance* and the *performance due to parallelism*. The first one is measured in units of time and the second by using the concept of *speedup*.

Since the goal of parallel processing is to reduce the execution time, we shall compare the execution times for different number of processors in order to study the accelerating effect of parallel processing. It may be expected that if the computations can be carried out completely in p equal parts, the execution time is nearly $1 / p$ of that for execution on only one processor. However, it is clear from Amdahl's law that the serial parts may have a negative influence on this reduction. The speedup S_p for a job running on a system of p processors is defined by

$$S_p = \frac{t_1}{t_p},$$

where t_i is the execution time for the job running in just one processor, and t_p is the execution time for the same job running on p processors. Special care must be taken when selecting the sequential version to be used in this process, because sequential algorithms with poor performance can lead to high (and even superlinear) speedups, resulting in parallel versions with good speedup, but still poor raw performance.

The model described above represents an idealized and simplified situation. In practice, the negative influence of communication and synchronization between processing elements, the increase in computational complexity that normally characterizes a parallel version of the program, and the lack of scaling in the problem size that can meet a higher processing power often reduce the theoretical speedup. We can overcome this sequential bottleneck, which grants Amdahl's law a rather pessimistic approach in regard to speedups from parallelism, by removing the restriction of a fixed problem size and using a fixed-time concept instead [Gustafson88].

For real-time signal processing applications, the goal of parallel processing is also to meet specified latency requirements. Therefore, the measure of the system *scalability* must take this factor into consideration by adopting a time-constrained scaling approach that can alleviate the sequential bottleneck and improve speedup by scaling the problem size with the increase in machine size.

The timing specifications for the RT_STAP benchmark, described in the Section 3.3, emphasize a similar approach, by determining the smallest machine size that is required to meet a prescribed real-time constraint by using (scaling) different problem sizes, algorithm complexities, and latency constraints. Note that this approach can also be viewed as scaling towards greater quality / accuracy of the results [Hwang98:136], since more complex STAP algorithms – with more powerful adaptive capabilities – are employed in the solution of the STAP problem.

Other alternatives for scalability analysis could have been the isoefficiency model [Grama93] and the isospeed model [Sun94]. The first is very little related to our problem domain, since maintaining a fixed speedup bears little relation to satisfying real-time constraints. The second model is more relevant, since keeping a fixed Mflop/sec speed translates in a measure of machine resource utilization as the number of processors increase. However, this approach does not constrain latency, which is of fundamental concern for real-time applications.

Two auxiliary measures of performance are used in this work for *specialized* purposes: *processing rate*, measured in floating point operations per second – Flops/sec, and *processor utilization*, measured in percentages of the peak theoretical capacity of the processing node. The former is used to characterize the performance of different implementations of the FFT, and some routines of the LAPACK package. The latter is used to convey the level of utilization sustained by some of the stages of the RT_STAP implementation when compared to the processor theoretical peak Flops/sec rate.

The performance metrics used to characterize inter-node communication performance are the *bandwidth* – measured in Mbytes/sec or Mbits/sec, and the *latency* – measured in seconds. Sometimes, the time spent in communication is also expressed as a fraction of the overall execution time of the program, for purposes of addressing the ratio between computation and communication for a program execution. Also, it is important to determine the overhead contributed by communication operations as a function of machine size.

In general, cost is an important metric for comparing machines. Although some form of cost-performance analysis is useful, they are measured separately, and cost is highly dependent on the marketplace [Culler98:228] and on the country's economy. Once cost is inserted as a factor, it is important to consider not only how performance increases with machine size, but also how cost increases in this process. [Hill95] pointed out that even if speedup increases much less than linearly, if the cost of the resources needed to run the program does not increase much more quickly than that, then it may indeed be cost-effective to use the larger (scaled) machine. In this thesis work, we address cost-performance of the ABC and AFIT NOW briefly in the last chapter.

4.4 Assumptions

We conclude this chapter with a discussion on the assumptions made. In this research, we consider that the packages downloaded - the FFTPACK and the ASCI Red Pentium Pro© BLAS – are reliable and function properly, although some correctness tests have been performed to check the effectiveness of some of their features. The same assumption on reliability and proper functionality is made in regard to software that is not supported commercially: the Linux operating system and the MPICH communication libraries. Finally, it is assumed that the RT_STAP implementation is correct, and designed in accordance to the specifications in [Cain97].

4.5 Summary

This chapter presented the full factorial design of experiments to be applied to the case study, and discussed the selection of factors and the number of replications. The section on statistical analysis highlighted the nature and purposes of the statistical tools to be used in this work. Finally, the section on performance metrics discussed an appropriate scaling model and the use of secondary performance measures. The next chapter – Results and Analysis – reports and explains the results of the experiments on the ABC, NOW, and IBM SP.

V. Results and Analysis

5.1 Introduction

In this chapter we describe the results of the experiments executed using the RT_STAP benchmark on the ABC, AFIT NOW and IBM SP. We start by analyzing the ABC communication performance delivered by the fast Ethernet interconnections and MPICH; then we refer to the analysis of the ASCI Red BLAS and of the FFT package selected. The remaining part of the this chapter concentrates on the analysis of the execution performance of different versions of STAP implementations, both sequential and parallel, in which explanations for the performance differences are provided according to experimentation, statistical analysis, and visualization of parallel program execution. The RT_STAP benchmarks are ported and executed on the AFIT NOW and the IBM SP, and the results obtained are also used to provide comparative insight on the performance of the programs running on the ABC.

5.2 MPI

All interprocess communications carried out by the parallel implementations of STAP algorithms in this thesis are based in the message passing paradigm and realized by means of the MPICH 1.1 software package, an implementation of the MPI standard loaded in the ABC. In this section, we empirically characterize the performance of the MPI implementation while running in the ABC's fast Ethernet switched network. The objectives are: (a) check how much bandwidth the ABC interconnection network can provide and compare the results to the maximal theoretical bandwidth, and (b) obtain quantitative information about the latencies incurred.

We executed a common communication benchmark, known as the *Ping-Pong* [Nupairoj94], and measured the average bandwidth and latency for messages of different sizes performing a round trip from processor zero to processor one and back, averaging over 100 messages for each size, from zero to 7200 bytes, on a two-by-two byte basis. The measurements for the bandwidth achieved are summarized in Figure 10 for two Pentium II 400 MHz processors.

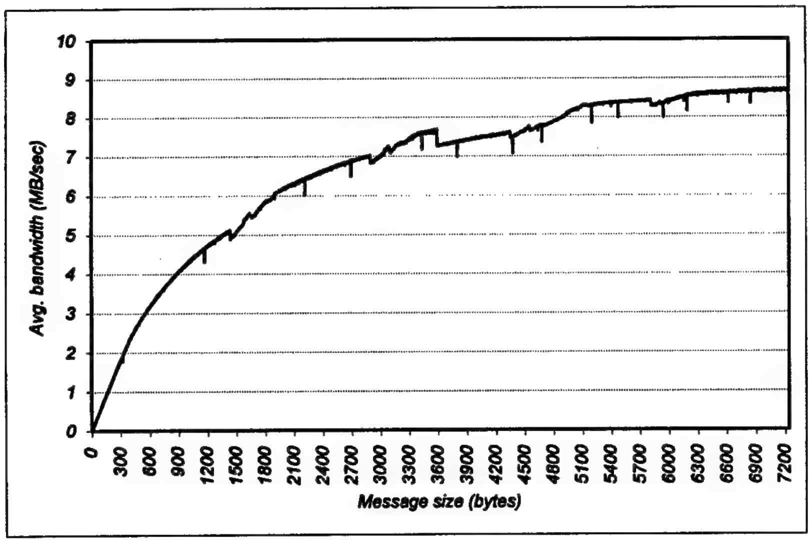


Figure 10 – Average bandwidth for different message sizes between two processors in the ABC using MPICH 1.1.

The result shows that the bandwidth achieved is directly proportional to the size of the messages, but the curve tends to flat close to 9 MB/sec for message sizes above 6000 bytes. In regard to latency observed, the corresponding graph is shown in Figure 11. The startup cost for a zero byte message is about 306 μsec, and the latency tends to increase linearly with the message sizes considered. The presence of occasional spikes in the latency are caused by the extra time needed by the TCP/IP protocol to acknowledge incomplete packages, and it is caused by a bug on the Linux 2.0.33 kernel (already corrected in the version 2.2 of the kernel).

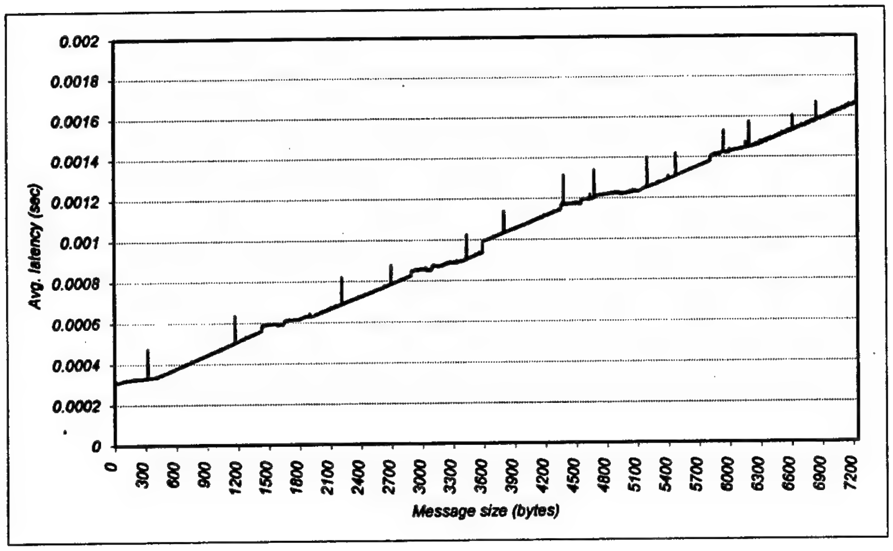


Figure 11 – Average latency for different message sizes between two processors in the ABC using MPICH 1.1.

We also performed similar experiments to stress the communication system by using considerably large messages, from 65536 bytes up to 32Mbytes, again registering the average bandwidth and latency observed, as depicted in Figures 12 and 13, respectively. In regard to the former, the observations show that the system is capable to capitalize around 86% of the theoretical peak capacity of the network, which is 12.5 MB/sec, for 16Mbyte messages; the latencies, though, tend to increase exponentially, requiring 0.2 seconds for a 1 Mbyte message.

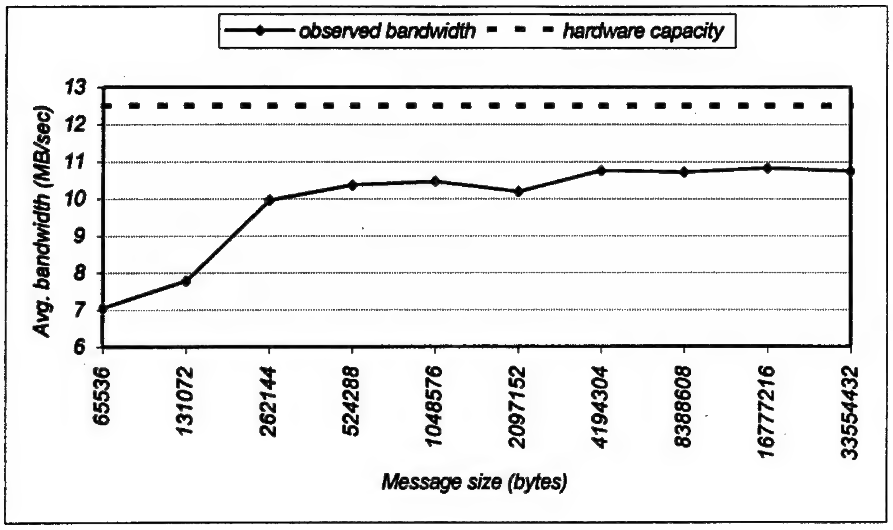


Figure 12 – Average bandwidth for large message sizes.

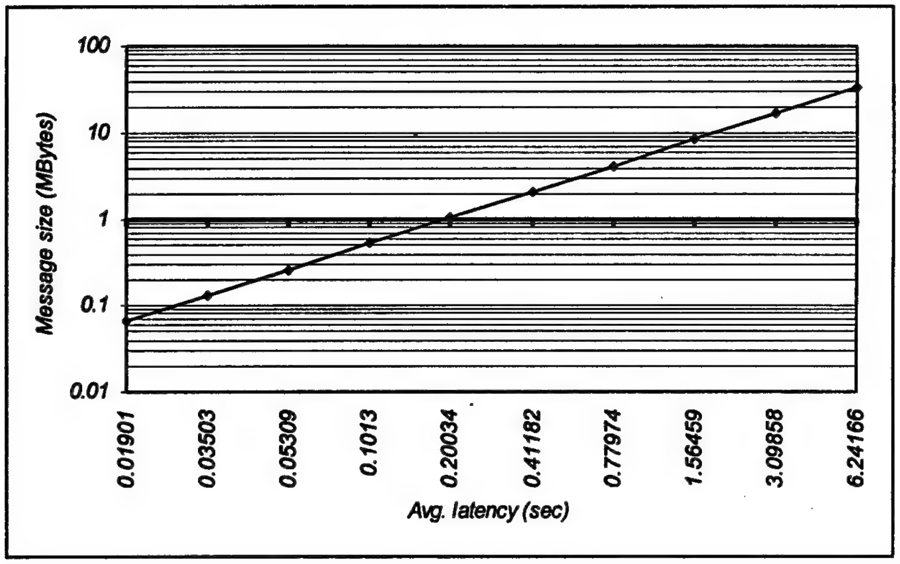


Figure 13 – Average latency for large message sizes.

These results confirm that the overhead for a single point to point communication operation on the ABC is high, and is long enough to perform millions of floating point operations. For the purpose of comparison, the startup cost for a point to point communication operation on the IBM SP2 multicomputer is 46 μs, and the average bandwidth for point to point communication is around 28.6 Mbytes/sec [Hwang96], by using a customized multistage communication network similar to a omega network, and the MPL proprietary communication library implementing the MPI standard.

As described in Appendix C, all MPI communication operations in the STAP implementations are carried out by collective operations. This fact brings other factor into play in regard to communication, which is the number of processors involved. The more processing nodes involved the higher the latency for the communication operation, especially due to synchronization. The Figure 14 below is a snapshot from the

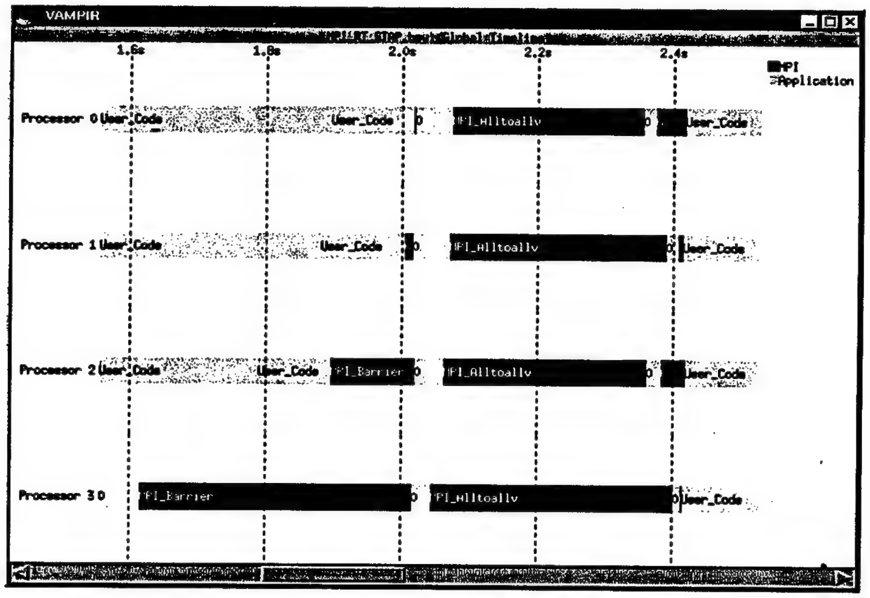


Figure 14 – Vampir timeline display showing an MPI collective operation involving four identical processors.

Vampir visualization tool for the second cornerturn operation in the first-order factored STAP implementation, involving 4 ABC Pentium 400 MHz processors. On that snapshot, we see that processors 2 and 3 have finished the portion of the task that is allotted to them but they are waiting in a barrier synchronization for processors 0 and 1 to finish their jobs. After synchronization, the local transposes and the packing of messages are executed, followed by the all-to-all operation. On the other end, more processing is required for unpacking and reorganization of data and subsequent synchronization for the next task.

In this example, we also note that even for a relatively uniform distribution of the load that exists in the STAP data parallel programs, we may see that the time spent in synchronization was pretty large compared to the time actually spent doing the all-to-all communication operation itself. That demonstrates the higher costs that synchronization operations may have in the ABC, apart from communication, for this specific case study. The average time required for barrier synchronization on ABC is listed in Table 10, as a function of the number of processors.

Number of processors	Avg. synchronization time (μsec)
2	179.06
4	357.18
6	665.89

Table 10 – Time required for barrier synchronization between 400 MHz processors on ABC.

Another important related issue is that RT_STAP also suffers from extra calculations and memory overhead to partition the data in a regular manner in order to facilitate the programming of communication (overhead due to parallellization). Since this task is accomplished at the beginning of the program execution by the root processor (rank = 0), the other processors are kept waiting in barrier synchronization during the amount of time needed for partitioning, as shown in Figure 15.

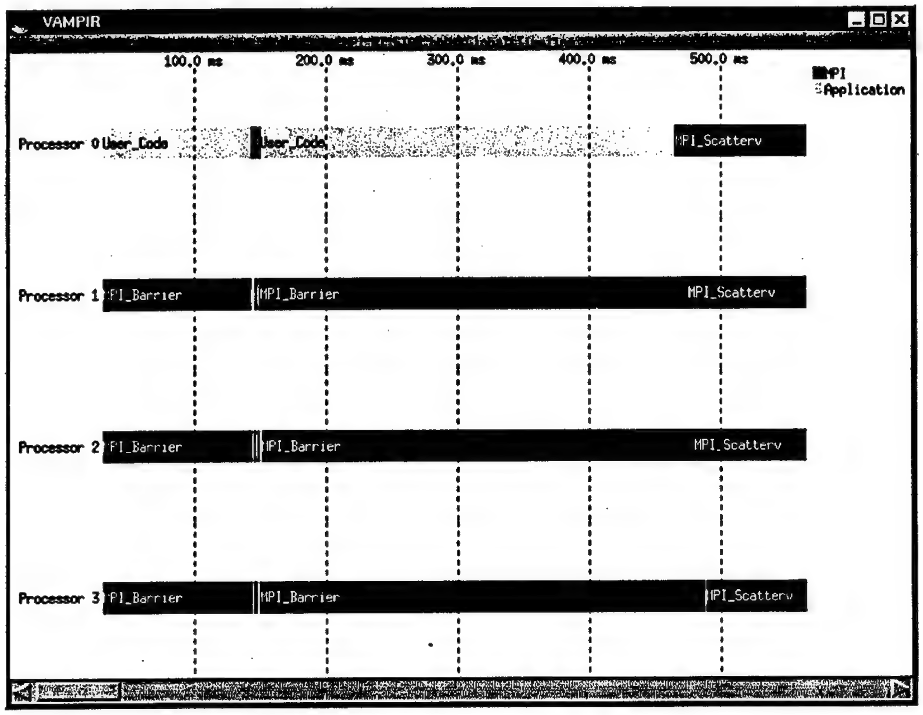


Figure 15 – Processor of rank 0 doing the initial processing in the RT_STAP while the other processors wait in a barrier synchronization.

Finally, we benchmarked the other most basic collective communication operation besides barrier synchronization, which is the *broadcast* operation. The results are summarized in Figure 16 for 2, 4, and 6 Pentium 400 MHz processors.

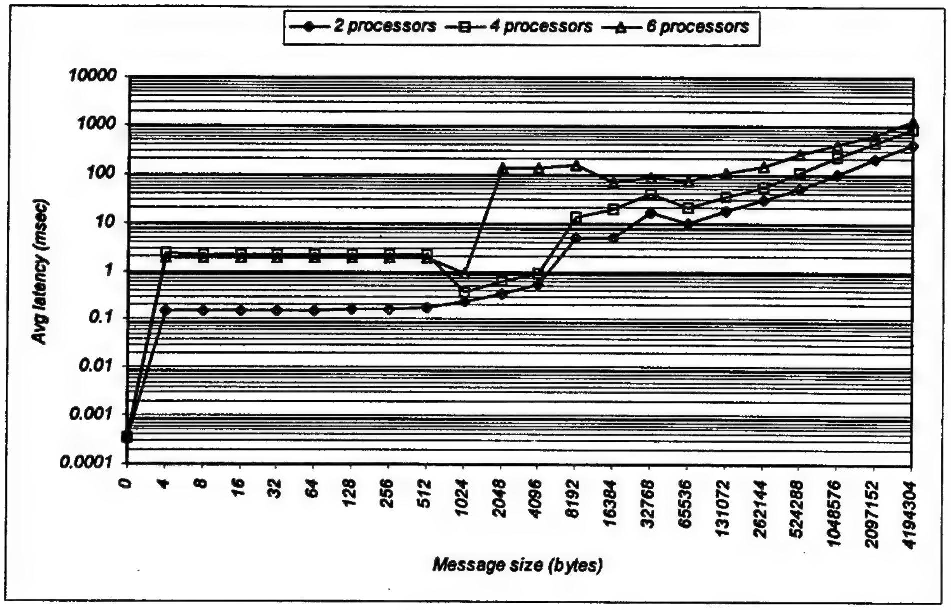


Figure 16 – Latency for the broadcast operation as a function of the message size and the number of processors.

Figure 16 tells us that the latency for small message sizes is fairly constant. As the message size gets larger, though, the latencies grow exponentially. The average time needed to broadcast a 0-byte message is 0.35μsec.

5.3 BLAS/LAPACK

We mentioned in section 3.2.4.1 the BLAS and the LAPACK packages. In that section, we described the selection and the installation of the ASCI Red Pentium Pro© BLAS on the ABC, as well as the building of two different LAPACK packages for testing purposes: one with a reference FORTRAN implementation of the BLAS and another using the ASCI Red BLAS. Here we discuss the results of the tests performed in regard to accuracy and efficiency. The first step was to execute the test programs for the BLAS and the two different versions of the LAPACK package. According to the outputs obtained, all the results of the correctness tests met or surpassed pre-defined thresholds and were archived for future reference in the appropriate subdirectories.

There are two distinct timing programs for LAPACK routines in each of the four data types (single and double precision real, and single and double precision complex, represented by the letters S, D, C, and Z, in this order), one for the linear equation routines and one for the eigensystem routines. The first set is also used to time the BLAS. From the set of input files provided we selected those of large size, which are appropriate for timing supercomputers and high-performance workstations [Dongarra94], both in single precision real and complex.

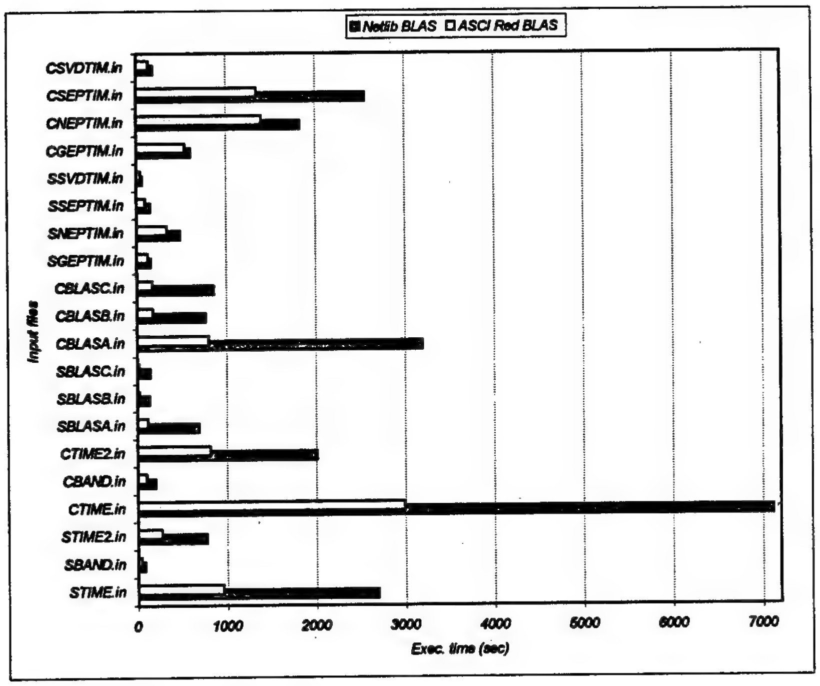


Figure 17 – Performance showed in the timing routines for two different versions of LAPACK.

The results of the timing test programs for the two different versions of LAPACK can be visually compared in Figure 17, according to the input file used. Table 11 shows the execution times and the improvements obtained on a Pentium II 333 MHz.

The first 12 input files are used to time the linear equation routines. From these twelve, the last 6 are special input files to time the BLAS. The remaining 8 files are used as inputs for the eigensystems routines. For more information on the organization of these files, like matrix types and dimensions employed, number of RHS, random number generation, block sizes and crossover points for blocked routines, and number of replications, the reader is referred to [Dongarra92].

Input file	ASCI Red BLAS	Netlib BLAS	Ratio
STIME.in	956.82	2697.41	0.355
SBAND.in	40.07	78.39	0.511
STIME2.in	266.51	774.13	0.344
CTIME.in	2988.83	7116.32	0.420
CBAND.in	100.94	199.62	0.506
CTIME2.in	813.91	2012.53	0.404
SBLASA.in	118.75	691.76	**0.172**
SBLASB.in	25.51	135.6	**0.188**
SBLASC.in	22.28	142.48	**0.156**
CBLASA.in	800.67	3195.1	**0.251**
CBLASB.in	177.24	766.4	**0.231**
CBLASC.in	171.95	861.71	**0.200**
SGEPTIM.in	120.5	157.35	0.766
SNEPTIM.in	338.48	485.39	0.697
SSEPTIM.in	97.31	151.96	0.640
SSVDTIM.in	46.52	64.63	0.720
CGEPTIM.in	540.3	607.49	0.889
CNEPTIM.in	1398.75	1828.42	0.765
CSEPTIM.in	1343.47	2558.99	0.525
CSVDTIM.in	136.47	191.95	0.711
Average ratio			**0.473**

Table 11 – Execution times in sec. for two different versions of the LAPACK package.

As expected, the performance of the ASCI Red BLAS was better than the simple reference implementation from the Netlib repository. On the average, the programs using the ASCI Red BLAS needed only 47% of the time needed by the reference implementation. Moreover, the timing tests for the BLAS showed even more dramatic reductions in execution times, as indicated by the ratios in bold at the rightmost column of Table 11 A reasonable explanation for superior performance for the ASCI Red BLAS is the *surface-to-volume* effect [Dongarra98:74], i.e., it presents a better ratio of floating point operations to data movement by exploring locality, especially for matrix-vector, and matrix-matrix operations.

To get a more detailed insight on the performance difference between the two LAPACK versions used to test the BLAS, we collected timing data from a very important

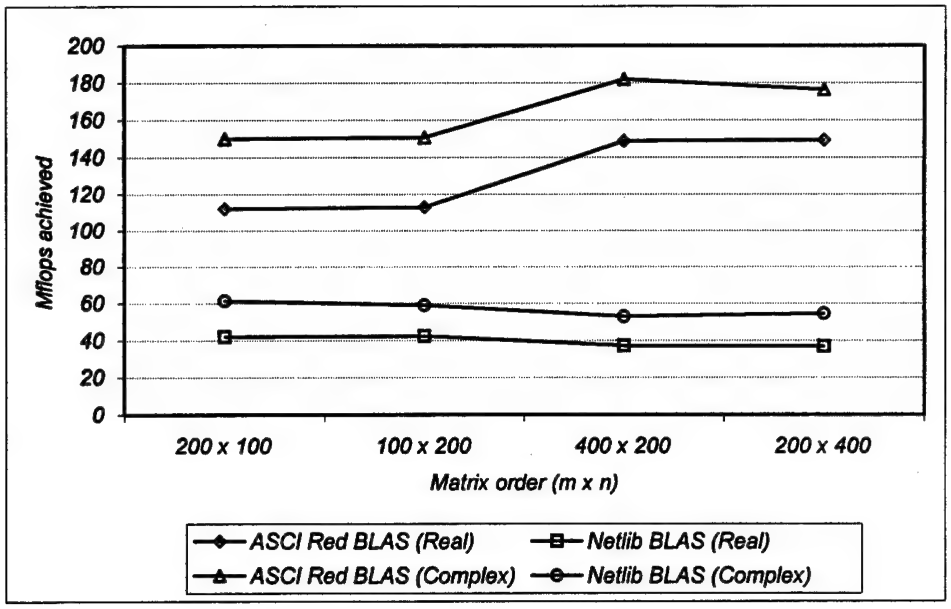


Figure 18 – Performance of the general QR factorization for rectangular matrices on one ABC processor.

mathematical operation that is widely used in science and engineering, the QR factorization of a matrix [Golub96:223]. Frequently, this routine is used to generate the least-squares solution of linear systems when $m \geq n$ and the $m \times n$ matrix A is of full rank. The results are presented in Figure 18 for rectangular matrices, and in Figure 19 for square matrices – both executed on a Pentium II 333 MHz. The use of Mflops as the performance measure in this case is consistent because these routines are provided with precise counts for floating point arithmetic operations, as listed in [Dongarra92:98].

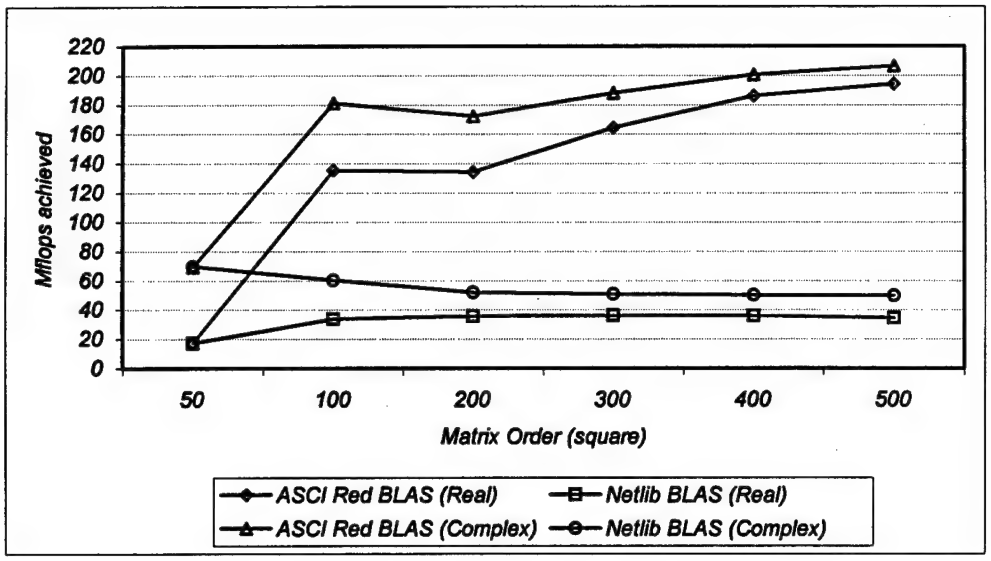


Figure 19 – Performance of the general QR factorization for square matrices.

The routines considered are SGEQRF and CGEQRF (for real and complex single precision numbers, respectively). They both perform blocked QR factorization with no pivoting based on the use of elementary Householder matrices [Golub96:224], in which the matrix Q is not formed explicitly. The parameters used in the algorithms were a block size NX = 48, and a crossover point NX = 128. This means that matrices with $n \leq 128$

should use the unblocked algorithm, and for $n > 128$ block updates should be used until the remaining submatrix has order less than 128.

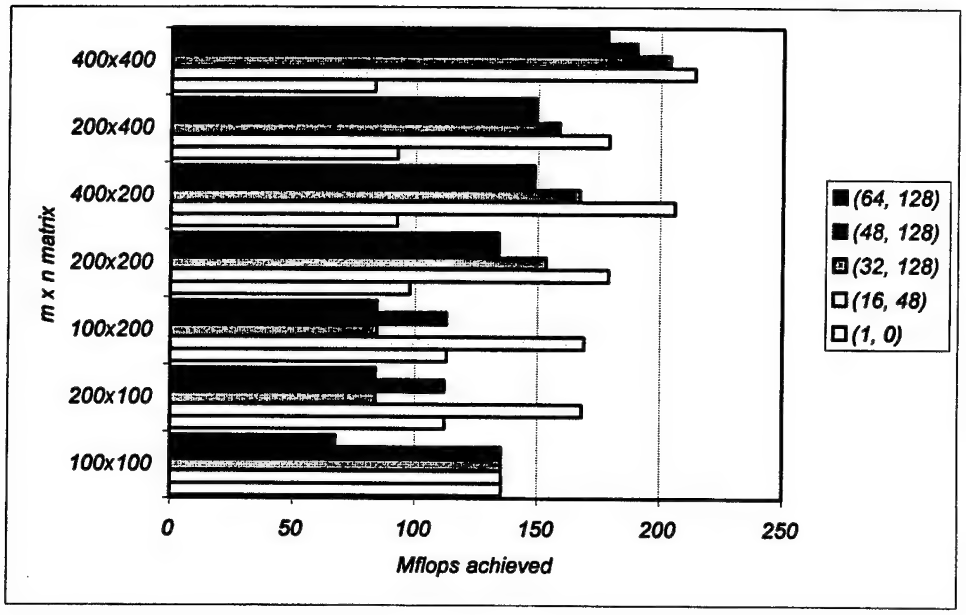


Figure 20 – Performance results for different values of the pair (NB, NX) for QR factorization on a Pentium II 333 MHz of ABC.

A closer examination on the performance results from the ASCI Red BLAS also indicate that good choices for (NB, NX) on the ABC can be 16 and 48, respectively, as exemplified in the Figure 20 for rectangular matrices of single precision real numbers. A value of NB = 1 means an unblocked algorithm.

5.4 FFT

In this section, we show the results for the experiments involving the fast Fourier transform to be used in the RT_STAP implementations, as discussed in section 3.2.4.2

The performance of the single precision one-dimensional complex transforms are summarized in Figure 21 for several different sizes that are powers of two, from 2 to 16384 array points, executed on a Pentium II 333 MHz.

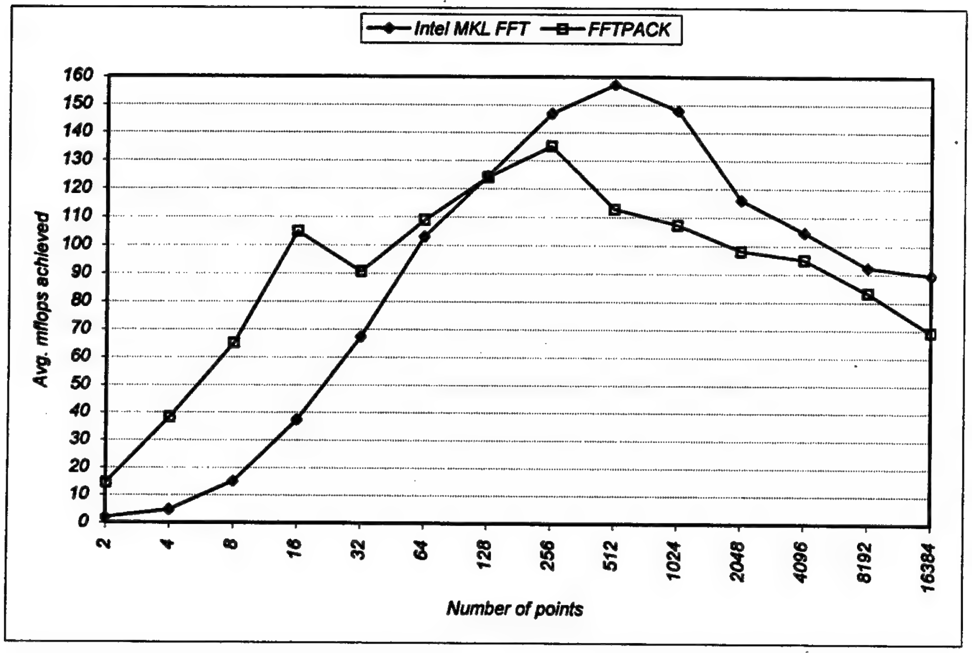


Figure 21 – Performance of two different FFT implementations executed on the ABC.

As mentioned in Chapter III, the FFT implementation from the Intel Math Kernel Library (MKL) was tested under the Windows NT 4.0 operating system, while the FFTPACK was built and tested under the Linux 2.0.33 environment. The fact that both implementations are based on the radix-2 Cooley-Tukey version of this algorithm allowed us to use the Mflops performance metric, since this version of the FFT accounts for *5nlogn* floating point operations [Golub96:190], and also because this metric is the one traditionally used in the literature for evaluating this algorithm – which allowed us to make other useful comparisons. The average Mflops values obtained for each size are averaged over a thousand replications for each array dimension, and special care was taken in not considering the first few executions to compensate for compulsory cache misses.

The graph shows that the FFT implementation from Netlib shows competitive results when compared to our reference MKL FFT implementation, especially for arrays up to 128 points. Considering that the sizes for the FFTs used in the RT_STAP implementation are 64 for the Doppler processing phase, and 256 for the overlap save method in the calibration/pulse compression phase, the FFTPACK can be expected to provide notable savings in execution time.

5.5 STAP

Up to this point, we presented and evaluated the performance of MPI, BLAS, and FFT in isolation, and commented on the effects that they may cause in the behavior of the STAP implementation. Now, we start to analyze the performance of the different STAP implementations by carrying out a side-by-side comparison between two versions of each different STAP program in the benchmark: the original version, from now on referenced as *original*, and the version that incorporates the FFT and BLAS routines, named *modified.* Another naming convention to be adopted is to call the first-order post-Doppler STAP simply by *FOPD*, and the high-order post-Doppler STAP as *HOPD*.

Initially, we describe the calculation of the number of replications (observations) needed for the experiments. Preliminary measurements showed that the FOPD running on more than 4 processors followed by the DPCA sequential program needed the highest values for n in equation 4-1 in section 4.2.1. We believe the reasons for this variability are: (a) because the FOPD implementation is less computationally intensive than the

HOPD, a greater percentage of its execution time refers to interprocess communication, which is a task that shows more timing variability than CPU computations; (b) execution times that, in absolute terms, are closer to the order of magnitude of the precision required - which is the case for DPCA processing - tend to originate large values of *n* in equation 4-1.

By applying the parameters from the FOPD running on five processors in the formula we estimate that approximately

$$n = \left(\frac{100 \times 1.960 \times 0.038764352}{(0.001 \times 1.763122 \times 100) \times 1.763122} \right)^2 = 597.375 \approx 600$$

600 replications are needed to get results within the desired precision and confidence. This value of *n* was used as an upper bound for all other experiments.

In order to save disk space when storing the different program outputs, to produce reasonably sized spreadsheets and charts, and to deal with all the experiments needed by the factorial design, we adopted the folowing strategy: we ran a given program for 20 times and calculated the the mean of these 20 execution times; then we repeated this process for 30 times, getting 30 values that are then finally averaged, for each factor combination needed. Next, we discuss the associated results for the sequential programs.

5.5.1 Sequential Implementations

This subsection shows the execution times obtained for the sequential versions of the DPCA, FOPD, and HOPD implementations in the RT_STAP benchmark, both in the original and modified versions. Figure 22 contains a graph showing the different execution times compared, and Table 12 includes the same elapsed times together with some descriptive statistics of the performance on a Pentium II 400 MHz.

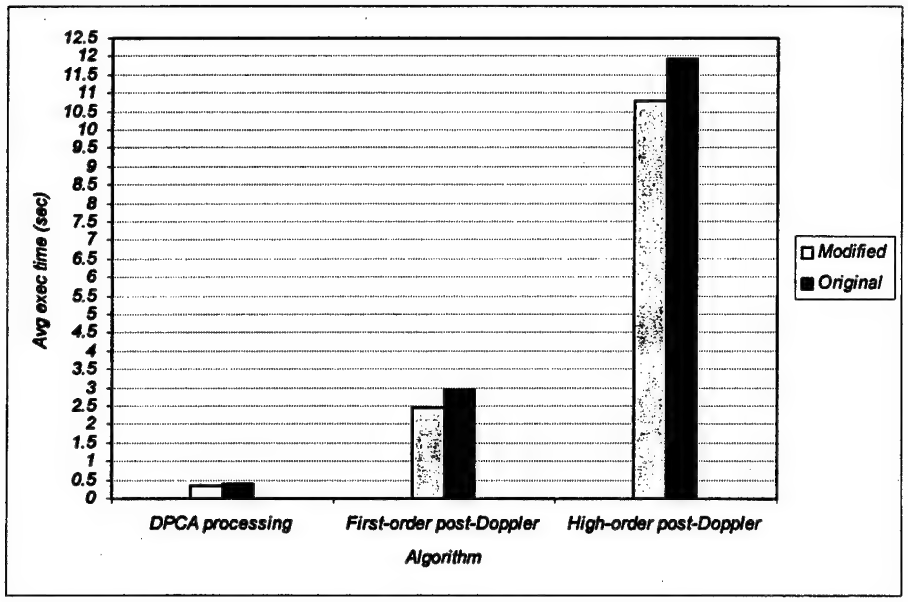


Figure 22 – Sequential performance of programs in the RT_STAP benchmark.

Version	Program	Avg. execution time	Standard deviation	95% Confidence intervals (+/-)
Original	DPCA	0.4070	0.0018	0.0007
	FOPD	2.9360	0.0014	0.0005
	HOPD	11.9090	0.0022	0.0008
Modified	DPCA	0.3530	0.0017	0.0006
	FOPD	2.4610	0.0007	0.0003
	HOPD	10.7720	0.0026	0.0009

Table 12 – Sequential performance in seconds, and associated descriptive statistics.

These results show improvements in sequential performance up to 16%. Better sequential performance is particularly important when one wants to consider speedups provided by parallel implementations. We constructed a Gantt chart for these sequential programs, as shown in Figure 23, to demonstrate how the elapsed time is partitioned among the different phases of the algorithms considered, including I/O.

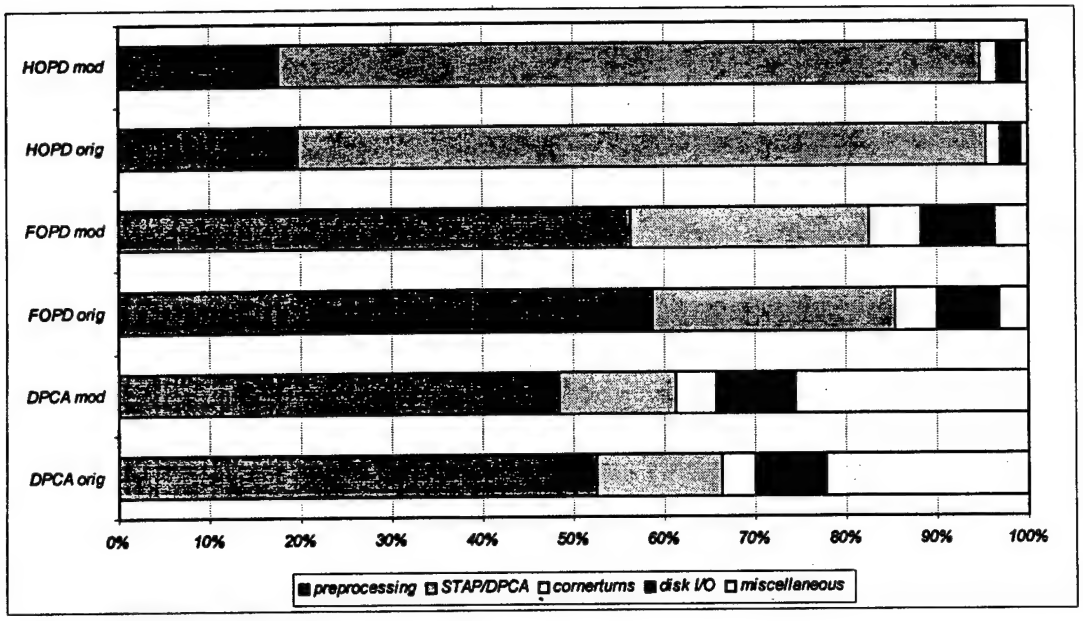


Figure 23 – Decomposition of execution times of sequential implementations.

The time that is accounted as miscellaneous is that relative to memory allocation/free time, generation of coefficients, and time spent to check the validity of input parameters. The item disk I/O encompasses time spent in reading the parameters file, the input data cube, the filter coefficients, and steering vectors. From the Gantt chart we can see that STAP processing dominates in the execution time of HOPD, and preprocessing is the phase that accounts for the largest portion of the execution time for FOPD and DPCA processing. Therefore, these two phases (preprocessing and

STAP/DPCA processing) are eligible to have their execution times improved through code modifications.

The use of descriptive statistics like means and standard deviations assume a normal distribution of the samples collected, as discussed in section 4.2.1. We validated this assumption for the samples collected and used in this subsection, by creating normal probability plots of the samples, like the one shown in Figure 24, for the original version of DPCA processing. As expected, the samples align themselves approximately along the straight line that represents a perfect normal distribution.

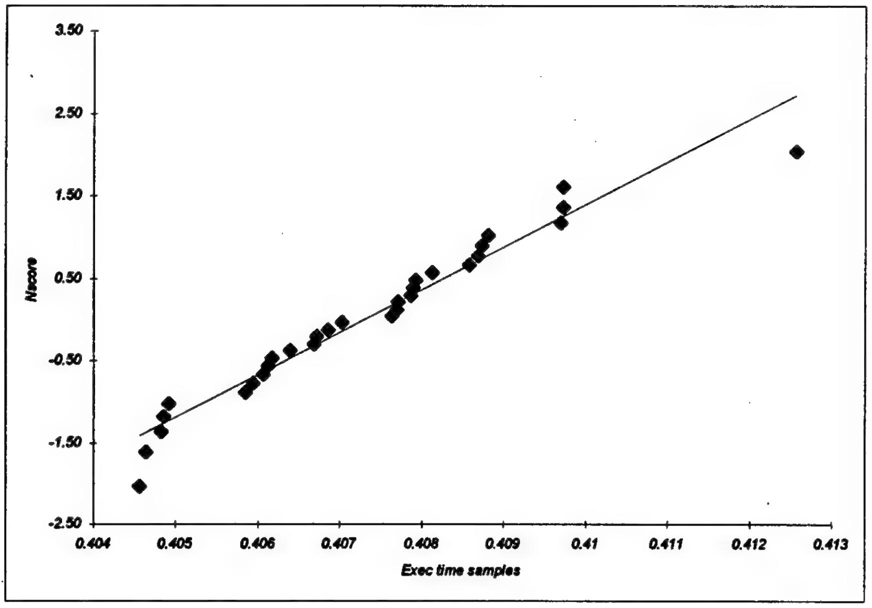


Figure 24 – Normal probability plot of samples collected for the execution times of DPCA processing.

Real-rime applications often rely on strict intervals for latencies, and the worst time taken for a given task to complete is important, as it is the mean time. As discussed

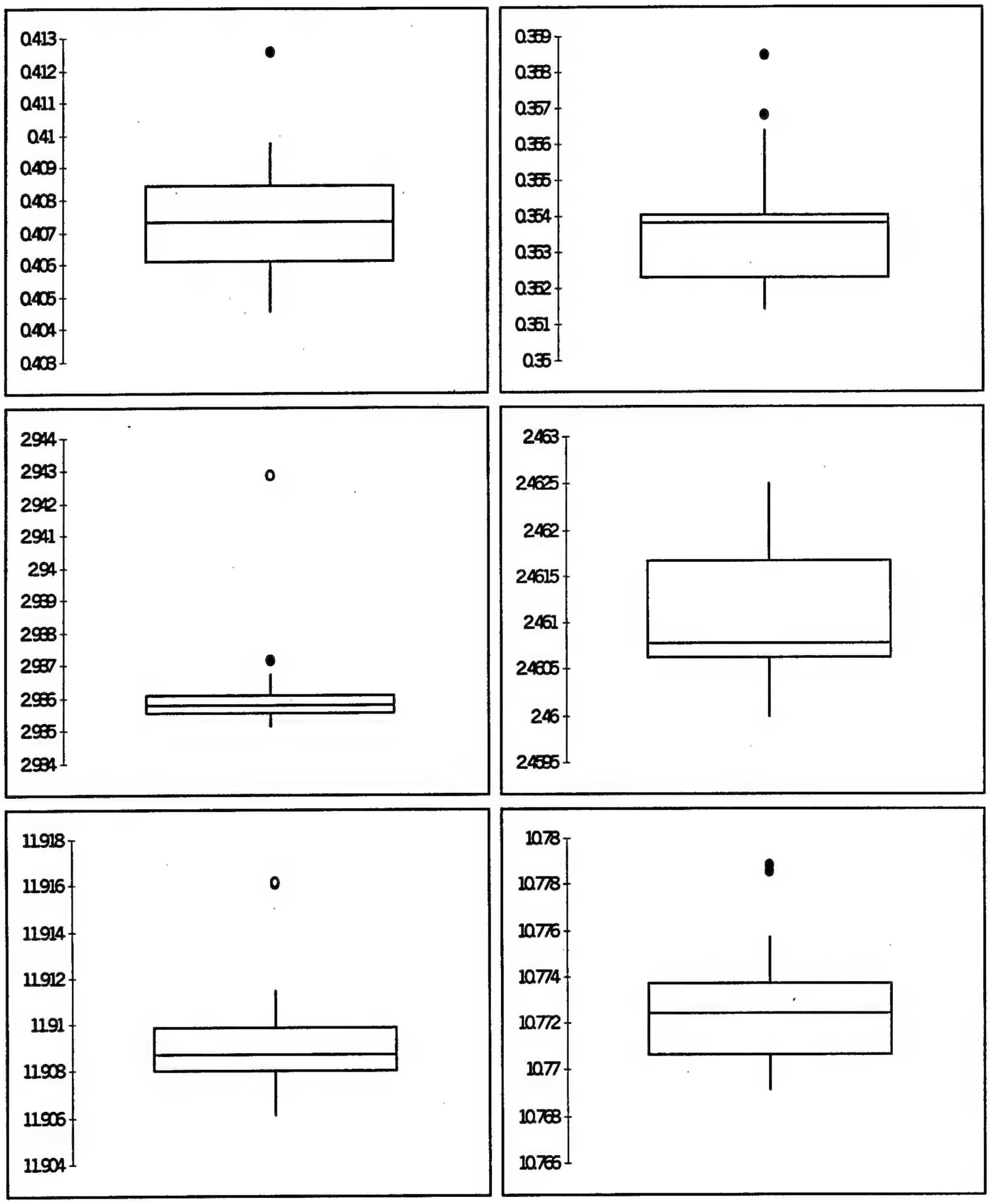


Figure 25 – Boxplots showing DPCA (top), FOPD (center), and HOPD (bottom) execution time samples, in the original (left col.) and modified (right col.) versions.

in section 4.2.1, the means derived should be pretty representative of time needed for the completion of the different tasks, provided outliers are not frequent in the collection of samples, since the ABC is a dedicated platform.

The use of boxplots allows us to check for outliers, as shown in Figure 25 for the three sequential versions. The plots show that outliers are rare and often moderate (represented by the black dots), i.e., they are concentrated within the limits defined by 1.5 to 3 times the fourth spread. This can add to our assumption of normality in the distribution of the samples, and dismiss the need for logarithm or square root transformations to reduce the skewness of the data.

We performed a two-factor ANOVA with replications between the factors *implementation* and *version* to examine: (a) to what extent the variation observed can be explained by the modifications made in the original source code; (b) the results of the F-test; (c) the significance of the changes in regard to the confidence level; (d) the hypothesis of equal means. The calculations were performed by using Microsoft Excel©, and are summarized in the Table 13.

The column *sum of squares* – SS – tell us about the variations. Obviously, the three program implementations explain much of the variation observed because they have very distinctive characteristics that lead to rather different execution times. The different versions of the program come in second place, being more representative than the variation that is due to errors and random fluctuation in execution times.

SUMMARY	Original	Modified				
DPCA processing						
Count	30	30				
Sum	12.21899	10.61578				
Average	0.407299667	0.353859333				
Variance	3.36314E-06	2.94881E-06				
First-order factored STAP						
Count	30	30				
Sum	88.08312	73.83317				
Average	2.936104	2.461105667				
Variance	1.87145E-06	5.13267E-07				
High-order factored STAP						
Count	30	30				
Sum	357.277691	323.18287				
Average	11.90925637	10.77276233				
Variance	5.03023E-06	6.80112E-06				
ANOVA						
Source of Variation	SS	df	MS	F-ratio	P-value	F crit
Implementation	4003.908395	2	2001.954198	585138229.4	0	3.047901487
Version	13.86000448	1	13.86000448	4051050.963	0	3.895451073
Interaction	8.941465128	2	4.470732564	1306721.472	0	3.047901487
Within	0.000595312	174	3.42134E-06			
Total	4026.71046	179				

Table 13 – ANOVA for the sequential programs.

All the *F-ratio* values are greater than *F-crit.*, indicating that the calculated *F-statistics* is in the upper 5% of the *f* distribution, and that the variance between factor levels is much higher than the variance that can be attributed to random error. The *P-values* are negligible, meaning that all the variations are significant at the 5% level, and that we can reject the hypothesis of equal means at this confidence level for any of the implementations.

We now report the results obtained by these sequential implementations in regard to the real-time requirements provided by the benchmark specifications. The data needed for the generation of the results comes from Table 6 (processing rates), Tables C.1 and C.2 (floating point operation counts) and Table 8 (parameters for each different implementation). Table 14 shows the percentage of the required Gflop/sec rate that the different implementations and versions were capable to sustain.

Sequential Implementations	Flops Count	Exec. Time original (sec)	Exec. Time modified (sec)	Benchmark Requirement (Gflops/sec)	% sustained original	% sustained modified
post-Doppler adaptive DPCA						
Video to I/Q conversion	7,127,040	0.105	0.103	0.22	30.85	31.45
Array calibration and Pulse comp.	8,454,144	0.097	0.057	0.26	33.52	57.05
Doppler processing	1,966,080	0.030	0.019	0.06	100.00	100.00
Weights computation	995,328	0.010	0.010	0.03	100.00	100.00
Weights application	491,520	0.006	0.006	0.02	100.00	100.00
first-order post-Doppler						
Video to I/Q conversion	57,016,320	0.853	0.842	1.77	3.78	3.83
Array calibration and Pulse comp.	67,633,152	0.778	0.458	2.10	4.14	7.03
Doppler processing	15,728,640	0.251	0.161	0.49	12.79	19.94
Weights computation	63,700,992	0.445	0.393	1.98	7.23	8.19
Weights application	3,932,160	0.037	0.037	0.12	88.56	88.56
high-order post-Doppler						
Video to I/Q conversion	78,397,440	1.158	1.145	2.43	2.79	2.82
Array calibration and Pulse comp.	92,995,584	1.067	0.631	2.88	3.03	5.12
Doppler processing	21,626,880	0.346	0.221	0.67	9.33	14.61
Weights computation	1,074,991,104	8.189	7.628	33.33	0.39	0.42
Weights application	16,220,160	0.143	0.138	0.30	37.81	39.18

Table 14 – Percentages of the Gflop/sec rates sustained by the different sequential implementations of RT_STAP.

The data presented in Table 14 shows an average 70% improvement in the sustained Gflop/sec rate for array calibration and pulse compression, 55% improvement for Doppler processing, and 10% improvement for weight computation, for the three sequential implementations. Additionally, an interesting inference can be made from the data displayed. Consider, for example, the percentage sustained for Doppler processing in the modified version of HOPD, which is 14.61%; a hypothetical data-parallel implementation without communication overhead would need a machine comprised of 100/14.61 ≅ 7 processors to be able to sustain the required rate for this stage. Obviously, this linear relationship does not hold, as discussed later in this chapter.

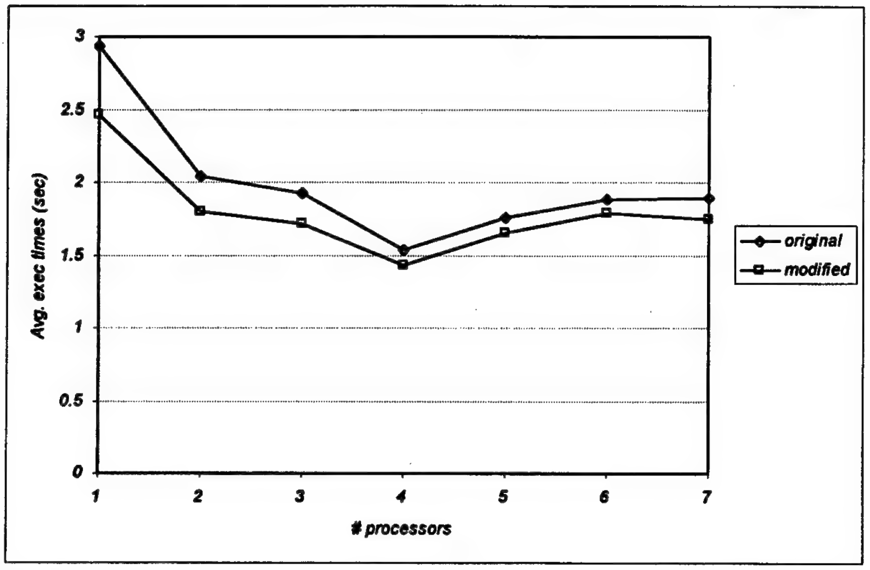
5.5.2 Parallel Implementations

This subsection presents the results of the experiments that deals with the concurrent implementations of the RT_STAP benchmark, and is further divided in two parts: one about the first-order post-Doppler, and the other about the high-order-post Doppler STAP. The analysis is done in much the same way as for the sequential implementations, except for the specific considerations in regard to concurrent programs and machines, like speedups due to parallelization and platform scalability.

5.5.2.1 FOPD

The first-order post-Doppler STAP represents an intermediate stage of scaled complexity and problem size, that is, higher accuracy is obtained by increasing the computational complexity while dealing with a larger problem at the same time. The focus of the analysis is centered around execution times, since elevating almost any other metric to this primary position runs the risk of favoring a parallel algorithm that runs slower over one that always runs faster. Close attention is to be paid to the communication performance as well, since this is the dominant factor governing scalability for parallel signal processing applications.

We show in Figure 26 the execution times and absolute speedup, followed by corresponding detailed statistical information in Table 15, obtained for FOPD running from 2 to 7 processors - 06 Pentium 400 MHz and 1 Pentium 450 MHz. We used the faster processor only when running with seven machines, and we believe that the difference in performance is not significant to discard the assumption of a homogeneous environment.



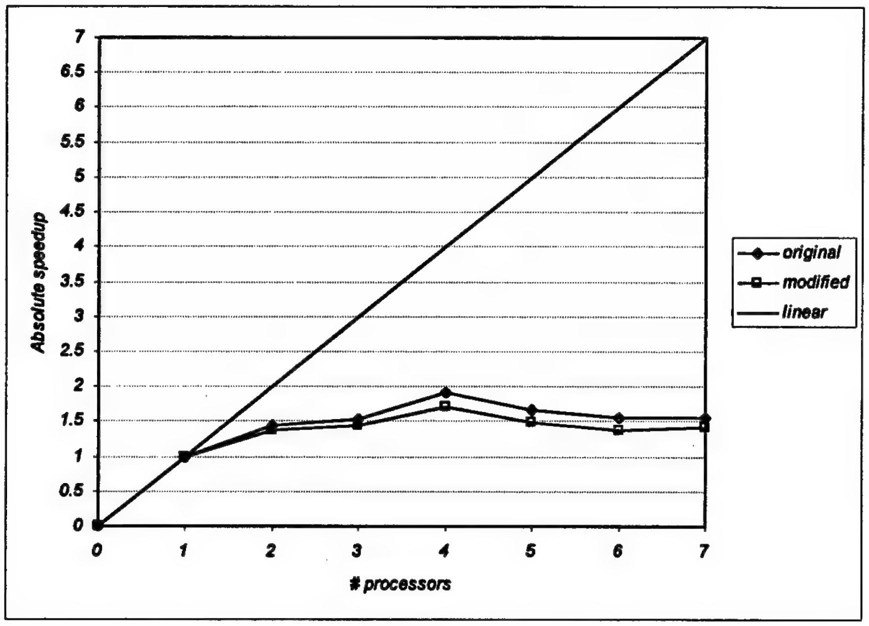


Figure 26 – Execution times and speedup for FOPD.

From the charts we can see that the program scales reasonably well up to 4 processors, when execution times start to increase because the latency of inter-process communication outperforms the reduction in computation times, affecting the program scalability. It is interesting to notice that although the modified version runs faster, its speedup is lower than the original version. This is an example of how misleading the measure speedup can be; in this case, a better sequential implementation is the explanation, and a choice based only in speedup would lead to a program with lower raw performance.

FOPD	number of processors	2	3	4	5	6	7
Original	avg. execution time	2.040	1.923	1.539	1.763	1.880	1.890
	standard deviation	0.003	0.007	0.014	0.039	0.034	0.062
	confidence interval (+/-)	0.001	0.002	0.005	0.014	0.012	0.022
Modified	avg. execution time	1.801	1.715	1.432	1.649	1.786	1.747
	standard deviation	0.005	0.007	0.011	0.026	0.024	0.049
	confidence interval (+/-)	0.002	0.002	0.004	0.009	0.009	0.017

Table 15 – Execution times and descriptive statistics for parallel FOPD.

A Gantt chart was built to describe how the execution times are spent among the different stages of the implementation, and the length of each bar represents the total elapsed time in seconds for each version and number of processors, as seen in Figure 27.

From the chart we can visualize that time spent in computation in the modified version reduces steadily with the increasing number of processors, but the cornerturns tend to dominate in the overall execution time, accounting for almost 1/3 of the elapsed time with 7 processors. Another portion that has a negative impact is source/sink collective communication time: *MPI_Scatter/MPI_Gather* from/to processor with rank 0 increases 216% from 2 to 7 processors. This is probably caused by endpoint contention [Culler98:154], that is, processor zero gradually becomes a hotspot in the MPI communicator.

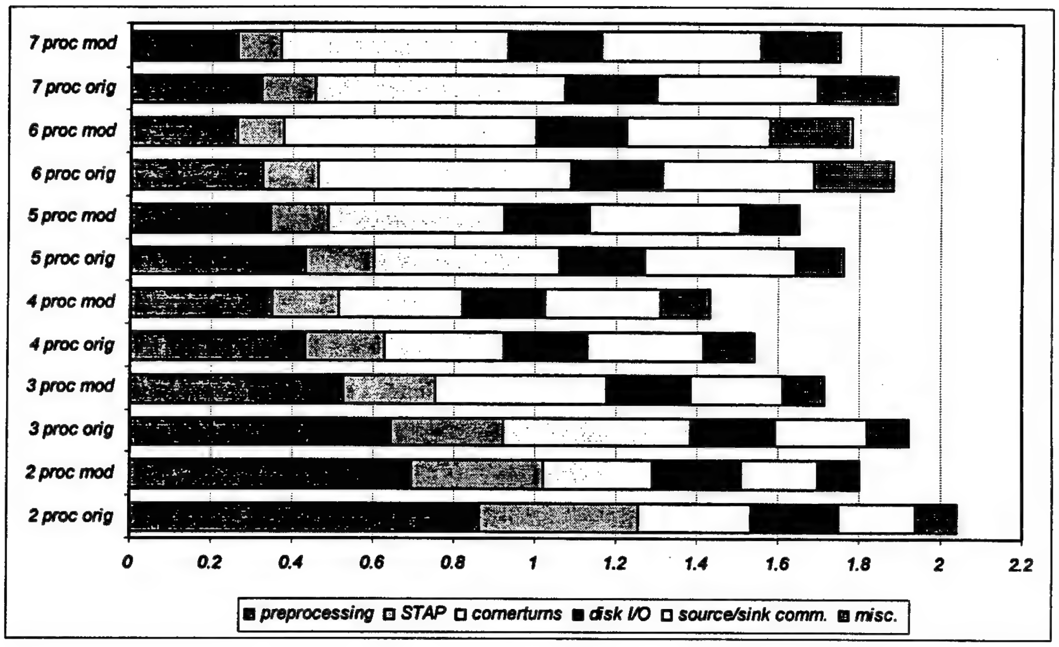


Figure 27 – Gantt chart showing the partitioning of the elapsed times for FOPD.

Likewise the sequential implementations, we now move to the statistical analysis of the data collected for FOPD. The assumption of normal distribution of the samples collected was checked, and a normal probability plot is shown for the original version running on five processors. Again, the samples align themselves approximately in a straight line in Figure 28.

Boxplots were generated for the modified version, and can be seen in Figure 29. Here, the larger variability of time spent in communication is apparent when we examine the plots for five to seven processors; the samples collected spread a larger range, and the outliers are more frequent and often severe. This happens because communication dominates a greater part of the elapsed time, instead of computation, and the execution times tend to vary according to the randomness of the network latency. This variability increases with the number of processors.

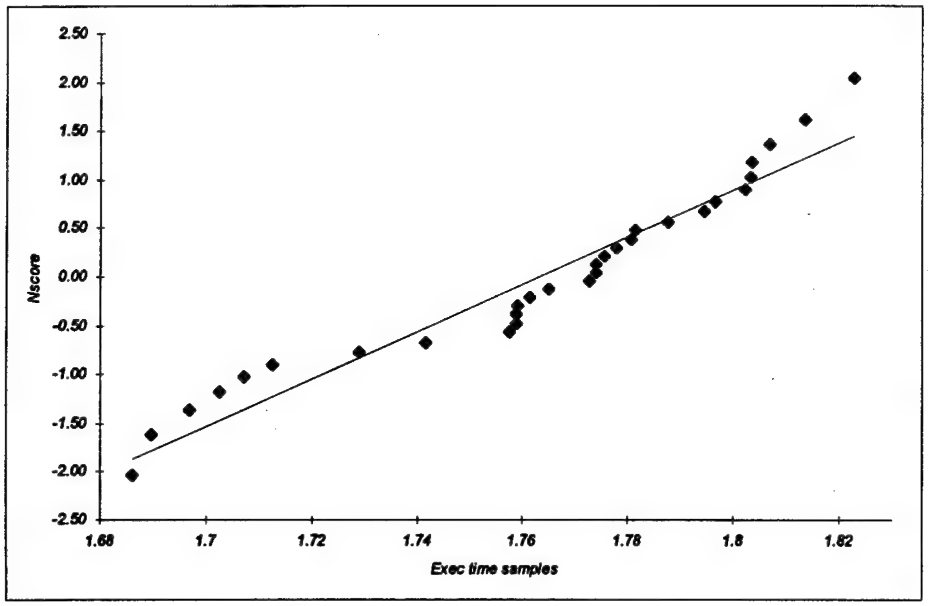


Figure 28 – Normal probability plot for execution time samples from FOPD running on five processors.

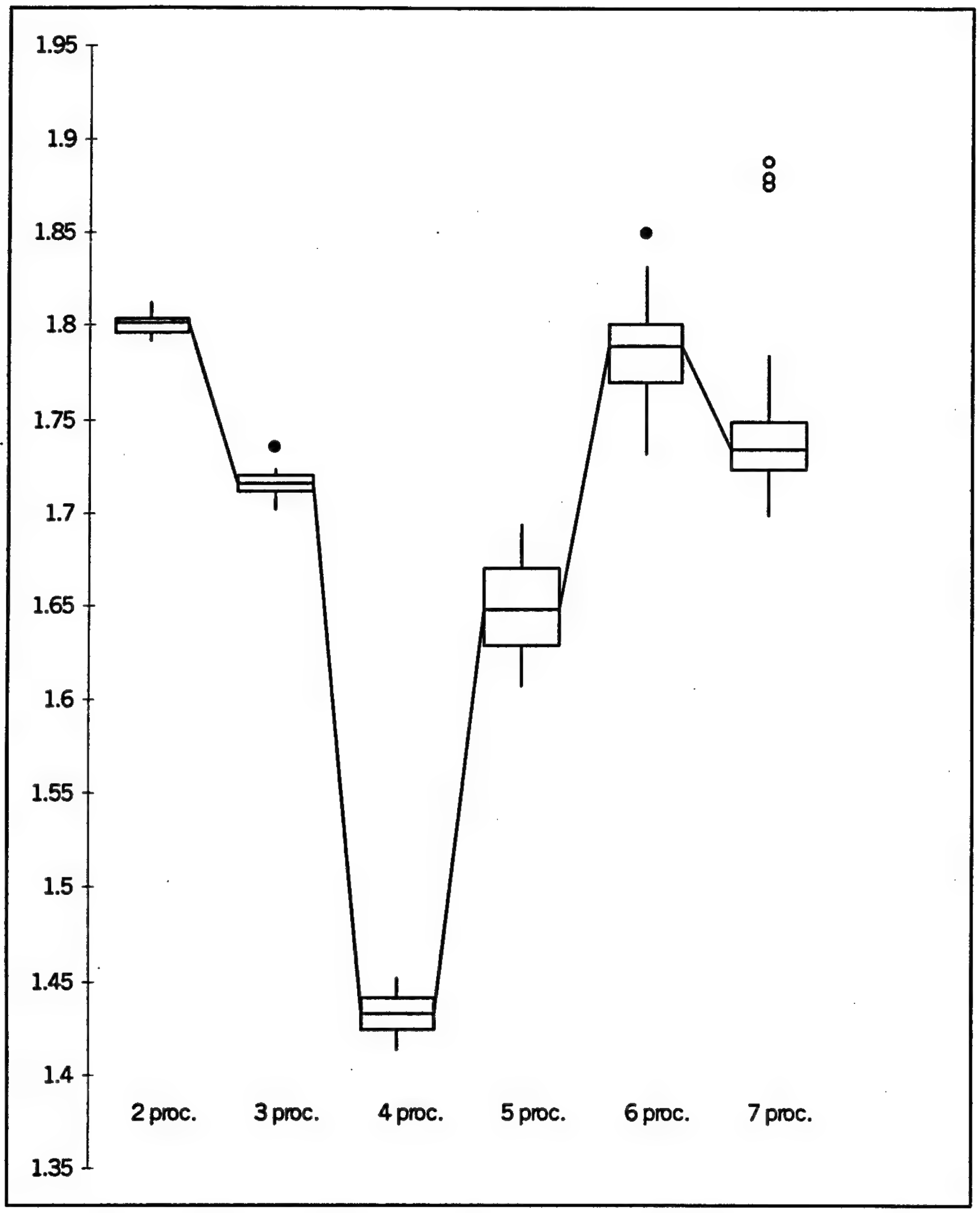


Figure 29 – Boxplots for the modified version of FOPD.

The next step is to perform ANOVA. The results of calculations are shown in Table 16. The parameters show that the modifications applied to the original version generate variations in execution time that are significant in the 5% level (all *P*-values are

SUMMARY	2	3	4	5	6	7
Original						
Count	30	30	30	30	30	30
Sum	61.20079	57.71298	46.19899	52.89366	56.40759	56.72093
Average	2.040026333	1.923766	1.539966333	1.763122	1.880253	1.890697667
Variance	1.04351E-05	4.48202E-05	0.0001829	0.001502675	0.00117865	0.003790608
Modified						
Count	30	30	30	30	30	30
Sum	54.035303	51.46373	42.96355	49.47301	53.58949	52.43909
Average	1.801176767	1.715457667	1.432118333	1.649100333	1.786316333	1.747969667
Variance	2.95559E-05	4.29797E-05	0.000128895	0.000681307	0.000583155	0.00237462

ANOVA						
Source of Variation	SS	df	MS	F-ratio	P-value	F crit
Version	2.050696054	1	2.050696054	2332.412286	2.4466E-156	3.868322551
Number of processors	6.965419253	5	1.393083851	1584.460009	6.311E-237	2.239929131
Interaction	0.263338768	5	0.052667754	59.90303412	6.6255E-45	2.239929131
Within	0.305967444	348	0.000879217			
Total	9.585421519	359				

Table 16 – Two-factor ANOVA for FOPD.

less than 0.05). The hypothesis of equal means for any of the experiments is rejected, and the variation due to random errors has much less significance than the factors being analyzed.

Concluding this section, we analyze the results according to the benchmark requirements. Table 17 summarizes these comparisons in the same way we did for the sequential implementations. The data refers to FOPD executing on 7 processors. The percentages sustained show that the FFT implementation was capable of meeting the requirements for Doppler processing. However, we

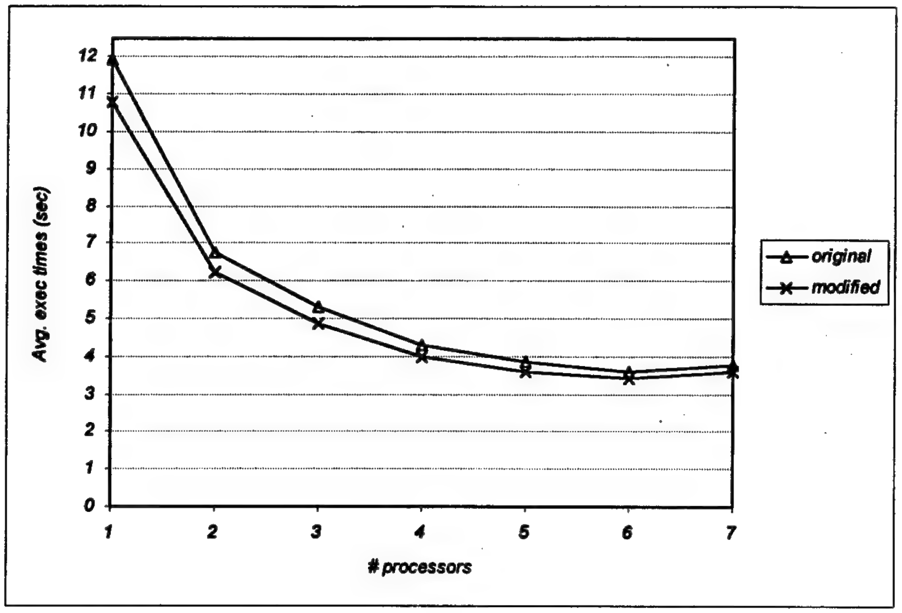
first-order post-Doppler	Flops Count	Exec. Time modified (sec)	Benchmark Requirement (Gflops/sec)	% sustained modified
Video to I/Q conversion	57,016,320	0.157	1.77	20.52
Array calibration and Pulse comp.	67,633,152	0.086	2.10	37.45
Doppler processing	15,728,640	0.030	0.49	100.00
Weights computation	63,700,992	0.062	1.98	51.89
Weights application	3,932,160	0.005	0.12	100.00

Table 17 – FOPD sustained performance according to benchmark specifications.

believe a faster implementation (possibly less portable) is needed in order to increase the sustained flops/sec rate for the preprocessing stages. An efficient signal processing library would be of help in this work, improving the performance of FIR filtering and decimation, as well as of the discrete linear convolutions. The ABC interconnection network did not show good scalability, and impaired the execution time improvements that more than 4 processors could have provided.

5.5.2.2 HOPD

The high-order post-Doppler corresponds to the hardest case between the benchmarks. It is a generalization of the algorithmic concept applied to FOPD, being more computationally intensive. The increase in complexity and problem size improves the speedup and the scalability of the adaptive processing. A better computation / communication ratio for HOPD also allowed the ABC to show better scalability. Figure 30 shows the execution times and the speedups for HOPD, as a function of machine size (kept at the same configurations used for FOPD processing).



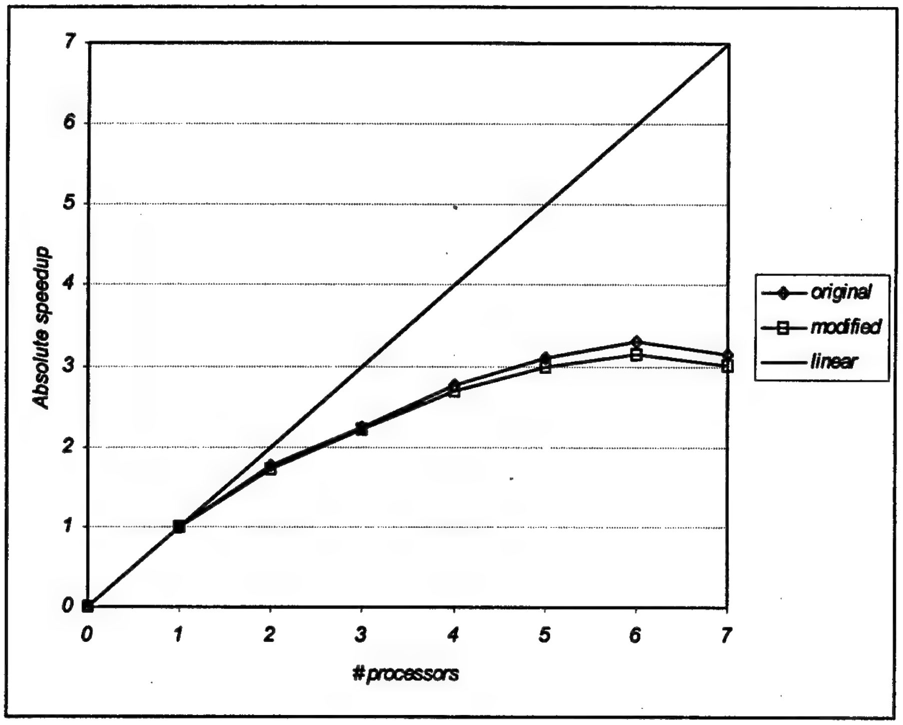


Figure 30 – Execution times and speedups for HOPD as function of machine size.

Again, the speedup of the modified version is slightly lower than that of the original implementation, although execution times were improved. This time, the program scaled well up to 6 processors, mainly because higher computation rates tend to provide better scalability in network computers with high latency in communication like the ABC. Table 18 provides the precise numerical figures for these executions.

HOPD	number of processors	2	3	4	5	6	7
Original	avg. execution time	6.771	5.315	4.295	3.846	3.600	3.792
	standard deviation	0.089	0.072	0.046	0.026	0.036	0.025
	confidence interval (+/-)	0.032	0.026	0.016	0.009	0.013	0.009
Modified	avg. execution time	6.236	4.873	4.002	3.604	3.416	3.585
	standard deviation	0.005	0.033	0.011	0.033	0.054	0.041
	confidence interval (+/-)	0.002	0.012	0.004	0.012	0.019	0.015

Table 18 – Execution times and descriptive statistics for parallel HOPD.

In order to observe how the elapsed times are partitioned among the different stages of HOPD, as well as time spent in communication, we also built a Gantt chart for this STAP implementation. The graph can be seen in Figure 31.

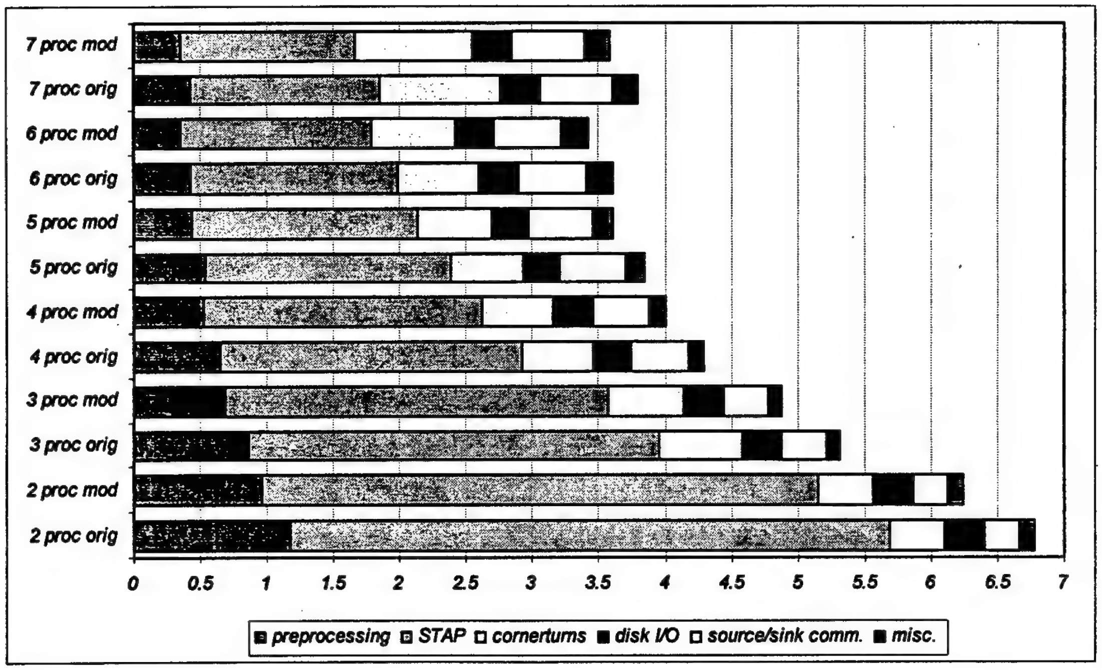


Figure 31 – Gantt chart showing the partitioning of the elapsed times for HOPD.

This chart shows a much more regular behavior of the program in regard to the number of processors, especially because the time spent in computation is dominant. Communication times due to cornerturns and source/sink communication cause the increase in execution times for 7 processors.

As we did with FOPD, we now move to the statistical analysis of the samples collected. First, the assumption of normality was again checked, and a normal probability plot for the original version of HOPD running on 7 processors is shown as an example in Figure 32.

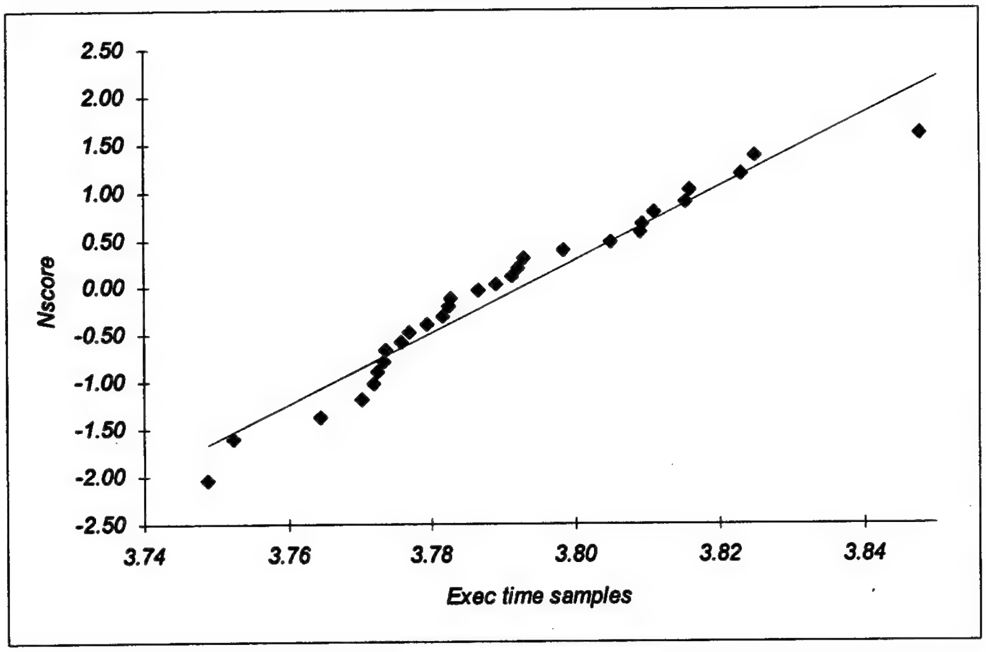


Figure 32 – Normal probability plot for HOPD running on 7 processors.

The distribution of the samples within the ranges for each number of processors and the existence of outliers can be seen at the boxplots built for the modified version in Figure 33.

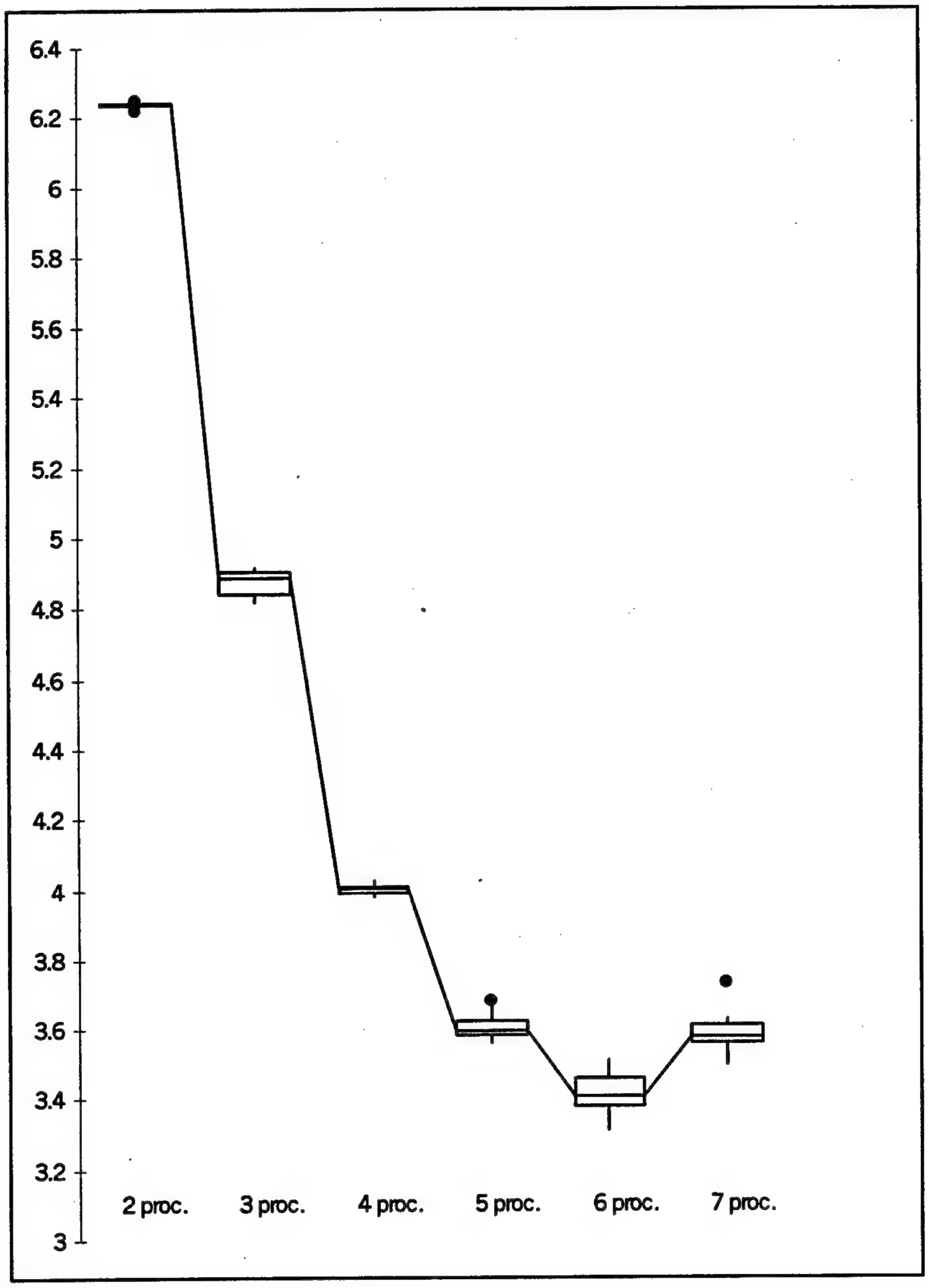


Figure 33 – Boxplots for HOPD.

The samples indicate a more regular distribution within their ranges, and no severe outliers are present this time. This is the result of more computation intensive algorithm that is less sensitive to the temporal variability of network communication.

The last statistical tool to be applied is the ANOVA, and the results are shown in Table 19. The observations made for FOPD are also valid for the results obtained here.

SUMMARY	2	3	4	5	6	7
Original						
Count	30	30	30	30	30	30
Sum	203.14573	159.46656	128.86583	115.40152	108.00608	113.76683
Average	6.771524333	5.315552	4.295527667	3.846717333	3.600202667	3.792227667
Variance	0.007881334	0.005121368	0.002094931	0.000666463	0.00129674	0.0006215
Modified						
Count	30	30	30	30	30	30
Sum	187.09613	146.20392	120.0637	108.13634	102.49256	107.57568
Average	6.236537667	4.873464	4.002123333	3.604544667	3.416418667	3.585856
Variance	2.76911E-05	0.001108411	0.000116111	0.001076257	0.002918274	0.001713618
ANOVA						
Source of Variation	SS	df	MS	F-ratio	P-value	F crit
Version	9.051689369	1	9.051689369	4407.807491	1.1092E-199	3.868322551
Number of processors	402.9324018	5	80.58648036	39242.36431	0	2.239929131
Interaction	1.489591531	5	0.297918306	145.0741942	7.45873E-83	2.239929131
Within	0.714638266	348	0.002053558			
Total	414.188321	359				

Table 19 – Two-factor ANOVA for HOPD.

In regard to the benchmark specifications, we see in Table 20 the modified HOPD was capable of meet the requirements only for the weights application stage, although it fell short for Doppler processing. For the latter, the linear relationship

high-order post-Doppler	Flops Count	Exec. Time modified (sec)	Benchmark Requirement (Gflops/sec)	% sustained modified
Video to I/Q conversion	78,397,440	0.208	2.43	15.51
Array calibration and Pulse comp.	92,995,584	0.115	2.88	28.08
Doppler processing	21,626,880	0.040	0.67	80.70
Weights computation	1,074,991,104	1.192	33.33	2.71
Weights application	16,220,160	0.022	0.30	100.00

Table 20 - HOPD sustained performance executing on 7 processors, according to the benchmark specifications.

between execution times and number of processors did not hold, as pointed out at the end of section 5.5.1. A considerably larger number of processors is needed to meet the flops/sec rate for the weight computation stage, and this fact demands a machine with excellent communication scalability in order to allow the adding of more processors without compromising the gains in computation times. In this regard, a high level of uniprocessor performance executing QR decomposition is a decisive factor.

One conclusion is that the ABC, relying on a high latency network (fast Ethernet), is not capable of scale up to a point where the weight computation requirements are met for HOPD. However, we believe it can be done on the ABC with no more than 20 processors for FOPD, if a faster interconnection network is used (Gigabit Ethernet, or Myrinet) along with a communication layer more efficient than TCP/IP.

Considering that the theoretical peak Mflops rate for the Pentium II 400 MHz is 400 Mflops (meaning an operation done at every clock cycle), the maximum utilization rates achieved for FOPD and HOPD were a reasonable 28% and 32% of the theoretical maximum, respectively. Although the interpretation of this metric depends a lot on the machine and the application, it is a good indicator of software performance tuning. The code from our case study was organized in modules that represent significant phases of the program's execution, which is desired from the point of view of obtaining meaningful timing measurements - but not necessarily to accommodate optimizations. Moreover, the cost of a library subroutine call was a performance issue when function call overhead was on the same (or greater) order of magnitude as the time required to the actual execution of the function. Because every function call involves some overhead, like extra variable management and stack pointer manipulation, the application can suffer from the repeated use of these routines, and care was taken not to employ library functions that caused this type of performance degradation.

5.5.3 Parallel Program Visualization

Statistical graphics and calculations are only as good as what goes into them from data collection. An ill-designed experiment or data collection process cannot be rescued by graphics [Tufte83:15]. Therefore, it is necessary to crosscheck the measurements whenever possible by using program execution visualization tools. Other useful purposes are exploration, description, and revealing of data and relationships between them.

In this section, we present results obtained through the use of the Vampir© visualization tool (http://www.pallas.de) for the modified HOPD program running on six Pentium II 400 MHz processors. The visualization tool was selected mainly because of the negligible overhead imposed by the tracing activity. Also, Vampir obtained the best overall average in an evaluation done in [Browne98] – which was partially sponsored by the DoD High Performance Computing Modernization Program – as described in the Table 21.

Tool	Robustness/Accuracy	Usability	Portability	Scalability	Versatility
AIMS	Fair	Good	Fair	Good	Good
nupshot	Good	Good	Good	Good	Good
Pablo Analysis GUI	Good	Fair	Fair	Good	Excellent
Paradyn	Fair	Good	Fair	Fair	Good
SvPablo	Good	Good	Fair	Good	Good
VAMPIR	Excellent	Good	Excellent	Excellent	Good
VT	Good	Good	No, platform-specific	Good	Good

Table 21 – Comparative analysis of visualization tools. Data from [Browne98].

According to the report, Vampir excelled in terms of scalability (number of concurrent processes that can be traced), portability (availability across the important parallel platforms) and robustness (quality of the user interface design). Figure 34 is a snapshot of

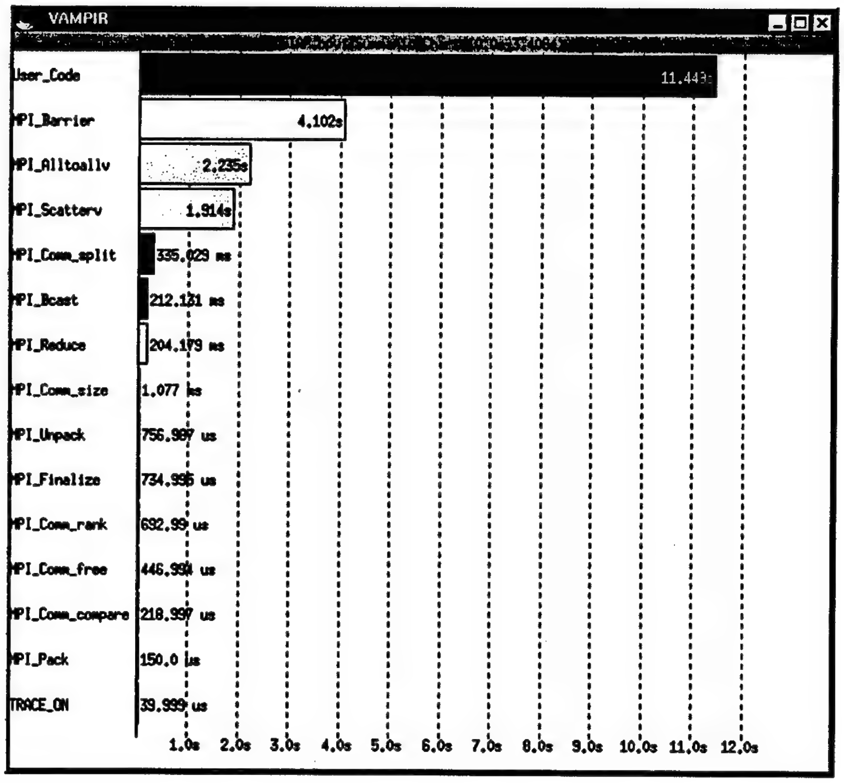


Figure 34 – Summaric chart of HOPD parallel program execution.

a horizontal histogram that classifies the time spent by all six processors performing HOPD code, as well as the different MPI primitives.

The graph tells us that the time spent in barrier synchronization is proportionally very large, and approximately equal to the sum of the times spent in the initial distribution of the data (Scatterv) and cornerturns (Alltoall). Also, the time referring to tracing activity is very small – less than 0.001% of the total time.

The next step is to show a global activity chart that could provide visual insight on the distribution of the tasks among the processors. We chose to display this information in the form of pie charts, as done in Figure 35. From the graphs, we see that processor zero (the root) shows a greater proportion of work done because it is responsible for the initial distribution of the data and final validation of the results. The

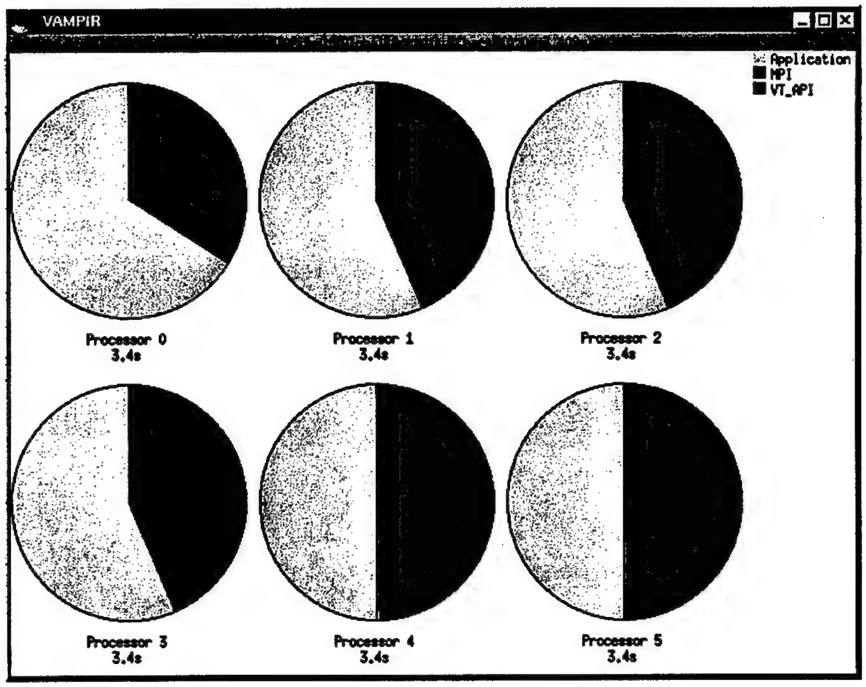


Figure 35 – Global activity chart for HOPD.

other processors show variable computation/communication rates, according to the variability of times spent in communication. The time relative to the tracing activity, represented in the legend by VT_API, is very small and does not show up in the chart. A close examination, done by zooming the activity chart over processors zero and one, tells us more about the influence of communication performance on HOPD. Figure 36

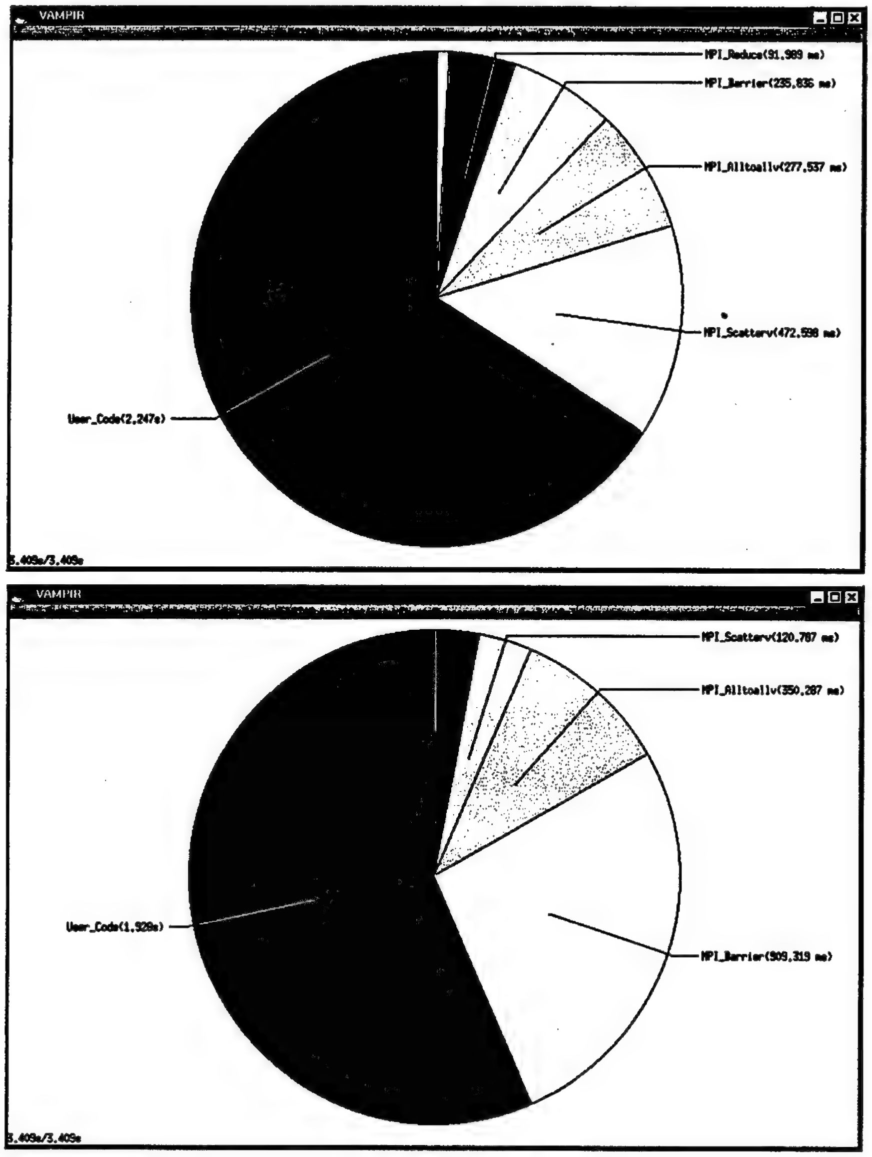


Figure 36 – Activity chart for processors zero (top) and one executing HOPD.

tells us the penalty incurred in terms of synchronization for non-root processors. While *MPI_Barrier* accounts for 6.9% of the time in processor zero, the percentage in processor one is around 26.6% of the total 3.409 seconds.

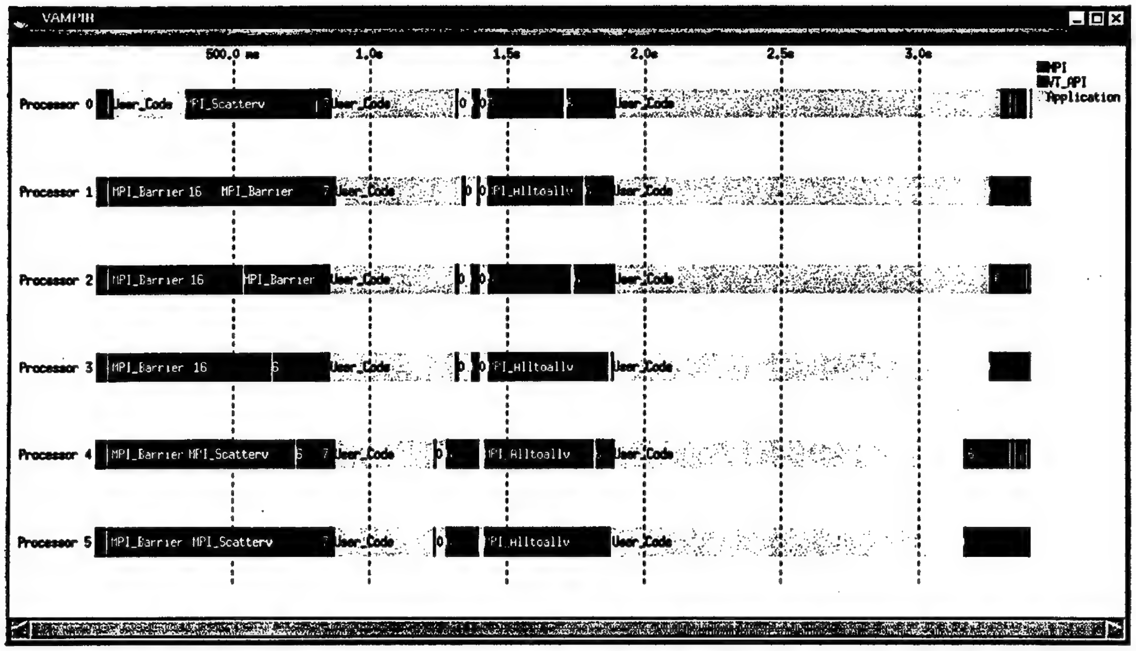


Figure 37 – Global timeline for HOPD execution.

Finally, Figure 37 shows the global timeline for the HOPD execution. The data parallel programming model is very apparent in this display, in which communication is concentrated at the end of the several stages of the program. Moreover, the proportion between computation and communication/synchronization times is easily visible. Again, time spent in tracing is negligible, and does not show up. The times reported by Vampir for the various stages of execution were comparable to the times collected by the experiments' timing routines, thus providing visual evidence of the accuracy of the elapsed times collected for the different stages of the programs.

5.6 Other Platforms: AFIT NOW and IBM SP

We ported, and examined the performance of the original version of RT_STAP in two additional platforms: the AFIT NOW (a cluster of 6 workstations) and the IBM SP (a MPP). The main purpose of these experiments was to observe the effect that different interprocess communication latencies and processor capabilities could have on both application and machine scalability. Because STAP is a much more computational intensive application, the execution times obtained from these two platforms were greater than those obtained by using the ABC - for the same number of processors (the workstations use 170 MHz Sparc processors, and the IBM SP uses 135 MHz processors), although the scalability results were different, and generally better. We start with the AFIT NOW. Figure 38 below shows the absolute performance for the three sequential versions of the package, followed by detailed information on Table 22.

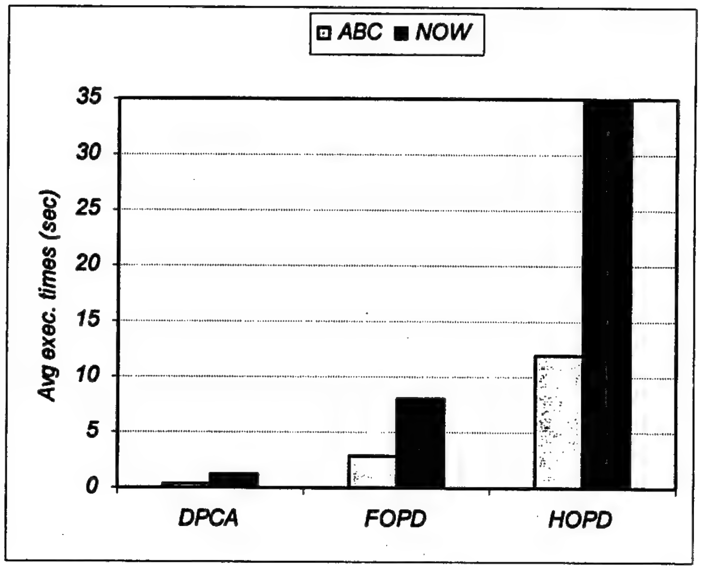


Figure 38 – Comparative performance of the STAP sequential implementations: ABC vs. NOW.

Implementation	NOW			ABC
	Exec. time NOW	Std. Dev. NOW	95% CI (+/-) NOW	Exec. time ABC
DPCA	1.166	0.003	0.004	0.407
FOPD	7.948	0.298	0.337	2.936
HOPD	34.799	0.283	0.320	11.909

Table 22 – Average execution times in seconds: NOW vs. ABC.

The execution times show that the sequential STAP programs run on average 2.8 times faster on ABC. We now concentrate on the parallel versions. Figures 39 and 40 show the execution times and speedups compared for FOPD and HOPD. Table 23 shows descriptive statistics for the NOW. The speedups obtained for both programs were better than those obtained on the ABC. Differently from ABC, the FOPD program had its execution times reduced when more than 4 processors were used, and the difference in

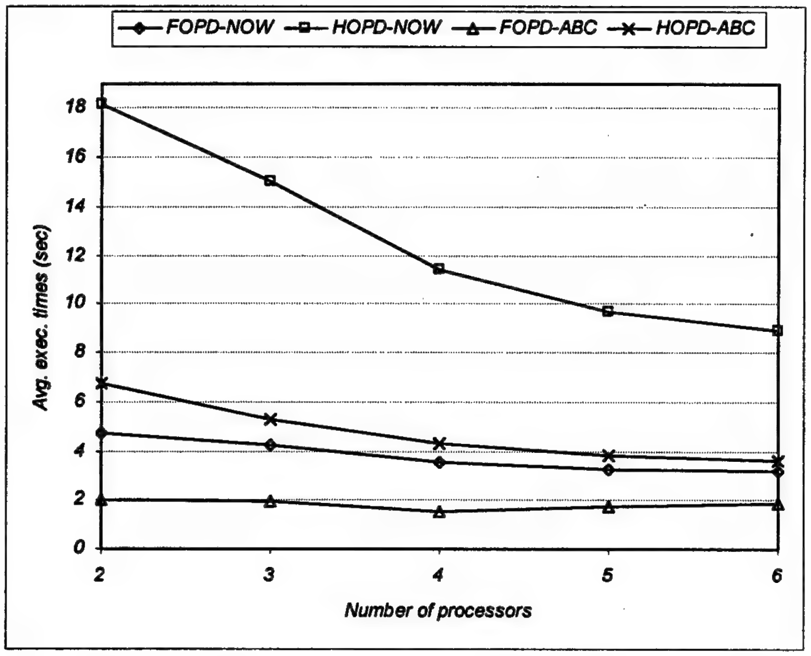


Figure 39 – Comparative execution times: NOW vs. ABC.

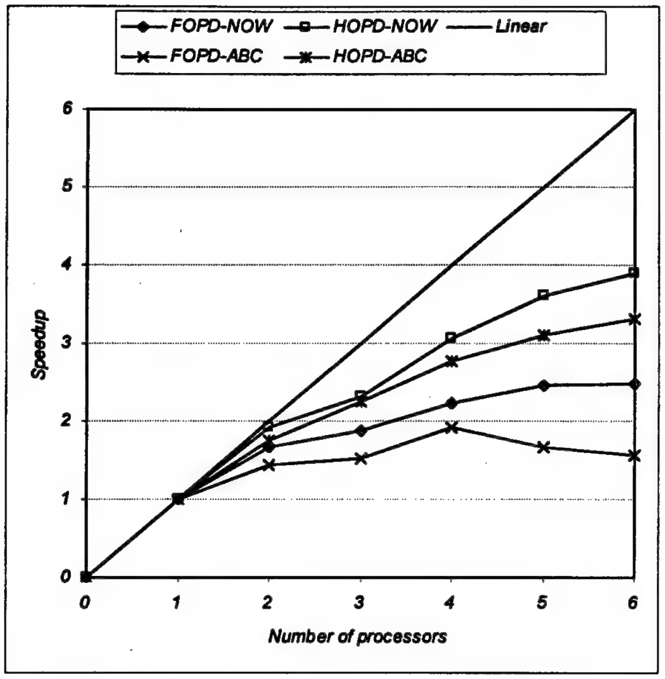


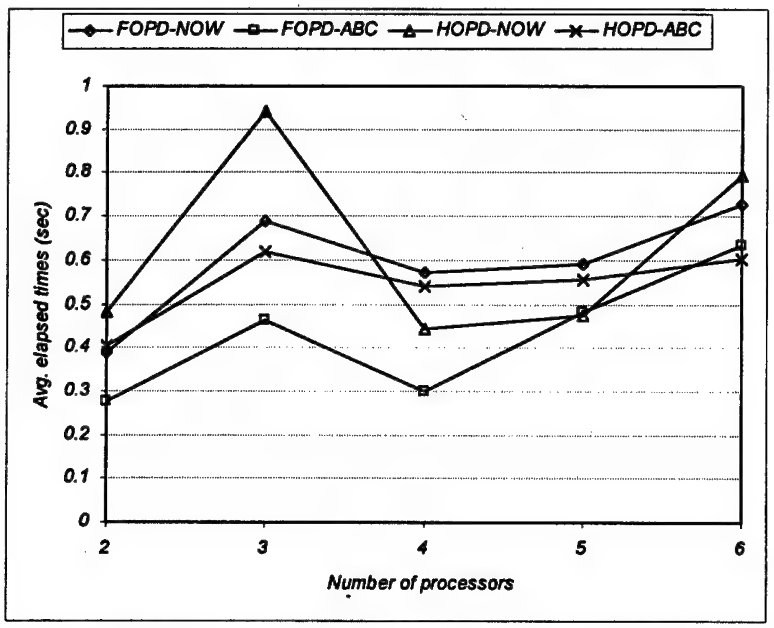
Figure 40 – Comparative speedups: NOW vs. ABC.

Program	number of processors	2	3	4	5	6
FOPD	avg. execution time	4.672	4.216	3.556	3.240	3.217
	Standard deviation	0.123	0.152	0.093	0.079	0.094
	95% confidence interval (+/-)	0.139	0.172	0.105	0.090	0.107
HOPD	avg. execution time	18.184	15.005	11.401	9.677	8.942
	standard deviation	0.291	0.112	0.220	0.147	0.126
	95% confidence interval (+/-)	0.329	0.127	0.249	0.166	0.142

Table 23 – Descriptive statistics for parallel RT_STAP programs on NOW.

speedups increased specially after 4 processors.

The reasons for these differences reside in better communication scalability and lower overhead provided by the pair Myrinet-TCP/IP. Although here the protocol is again the bottleneck that does not allow realization of better communication rates, we were able to get good results, specially for more than 3 processors. This occurred because the times spent in cornerturn operations and source/sink communication experienced improvements from 4 to 6 processors, as described in Figure 41. Specifically for more



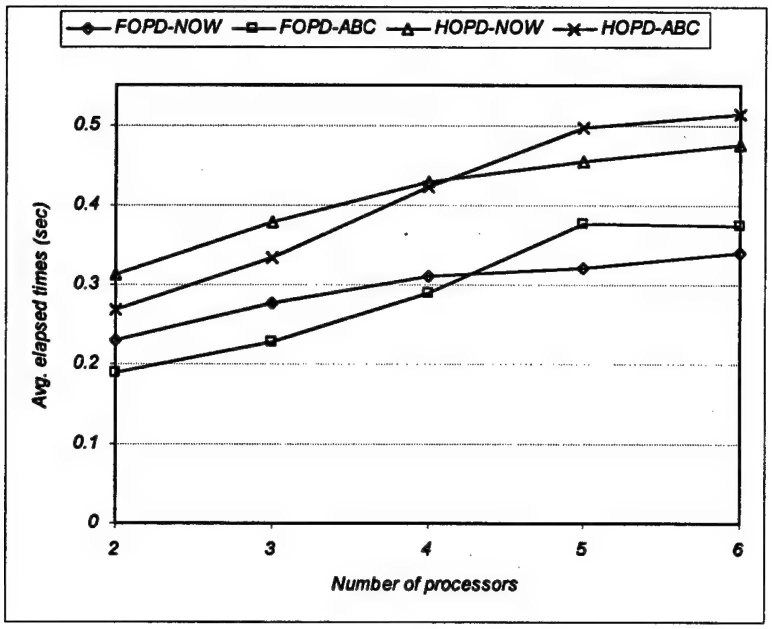


Figure 41 – Time spent in cornerturn operations (top), and source/sink communication (bottom), as a function of the number of processors: ABC vs. NOW.

than 4 processors, source/sink communication takes *less* time on the NOW.

The other platform used for comparison was the IBM SP located at the Aeronautical Systems Center Major Shared Resource Center (ASC MSRC), Wright-Patterson AFB. We start by reporting the performance of the sequential versions of the three implementations - DPCA, FOPD, and HOPD – in Figure 42 and associated Table 24.

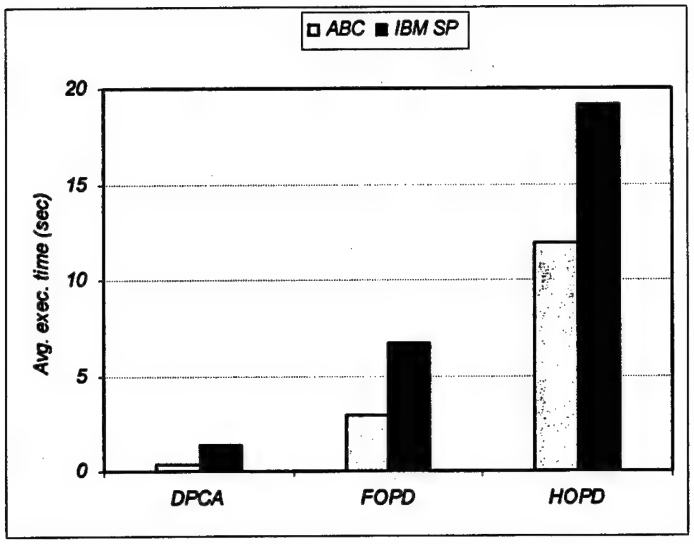


Figure 42 - Comparative performance of the STAP sequential implementations: ABC vs. IBM SP.

Implementation	IBM SP			ABC
	Exec. time IBM SP	Std. Dev. IBM SP	95% CI (+/-) IBM SP	Exec. time ABC
DPCA	1.401	0.0007	0.0007	0.407
FOPD	6.735	0.0031	0.0034	2.936
HOPD	19.202	0.2969	0.336	11.909

Table 24 – Average execution times in seconds: IBM SP vs. ABC.

The numbers show that ABC is on average 2.4 times faster than IBM SP when running the sequential implementations. The IBM SP demonstrated much more scalability in its interconnection network. We executed the parallel implementations of FOPD and HOPD using up to 64 processors (the limit imposed by the implementation),

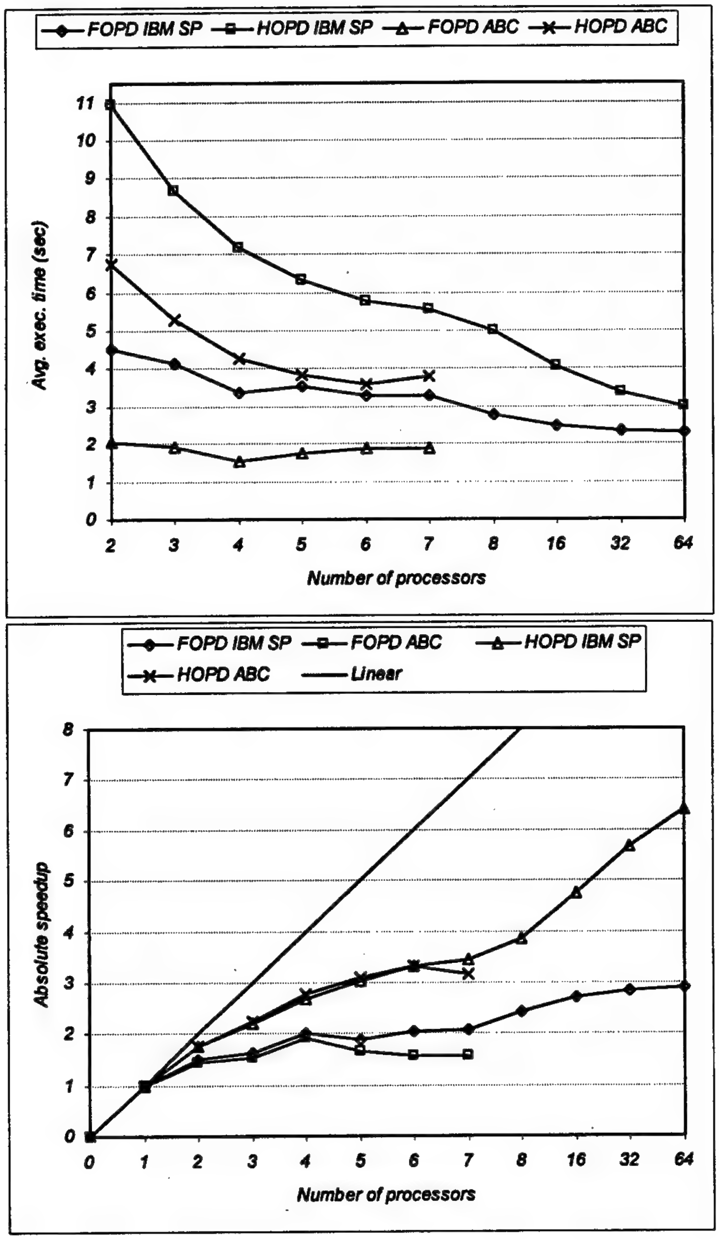


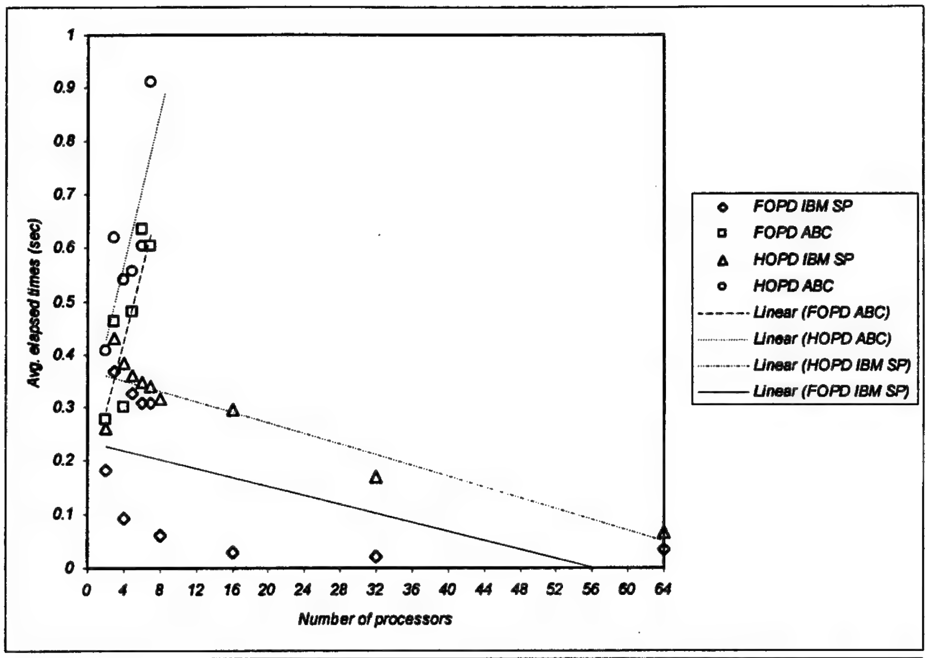
Figure 43 – Execution times and speedups for FOPD and HOPD on the IBM SP.

Program	number of proc.	2	3	4	5	6	7	8	16	32	64
FOPD	avg. execution time	4.518	4.156	3.365	3.561	3.283	3.272	2.790	2.488	2.361	2.329
	Standard deviation	0.010	0.002	0.001	0.002	0.0008	0.001	0.023	0.003	0.009	0.021
	95% CI (+/-)	0.011	0.003	0.001	0.003	0.0009	0.001	0.027	0.003	0.011	0.024
HOPD	avg. execution time	10.945	8.698	7.167	6.336	5.779	5.566	4.986	4.040	3.387	2.995
	standard deviation	0.028	0.034	0.023	0.002	0.035	0.001	0.018	0.002	0.031	0.017
	95% CI (+/-)	0.031	0.038	0.026	0.002	0.040	0.001	0.020	0.003	0.035	0.019

Table 25 – Descriptive statistics for parallel RT_STAP programs on IBM SP.

and we were able to get reductions in execution times with all processor counts. Figure 43 shows the charts for execution times and absolute speedups obtained on the IBM SP. The same data for the ABC is shown for comparison. Table 25 contains the descriptive statistics for IBM SP execution times.

From the execution time plot we can see that the better scalability of the IBM SP allowed us to use more processors and get competitive execution times, with sixteen processors or more, for HOPD. However, the FOPD implementation running on the IBM SP could not meet the performance obtained by ABC running with four processors, and the reason for that was I/O. The IBM SP spent more time in reading the input datacube, the parameters file, the filter coefficients, and the steering vectors. The sum of the time spent on these I/O tasks were on average 8.5 times higher than on the ABC (this average considers HOPD I/O times as well), and the effect of this higher latency was worse on FOPD because I/O ended up encompassing a larger part of its overall execution time, as the number of processors increased. Nevertheless, FOPD was able to show better performance than ABC in regard to the real-time requirements of sustained computation rates for the benchmark.



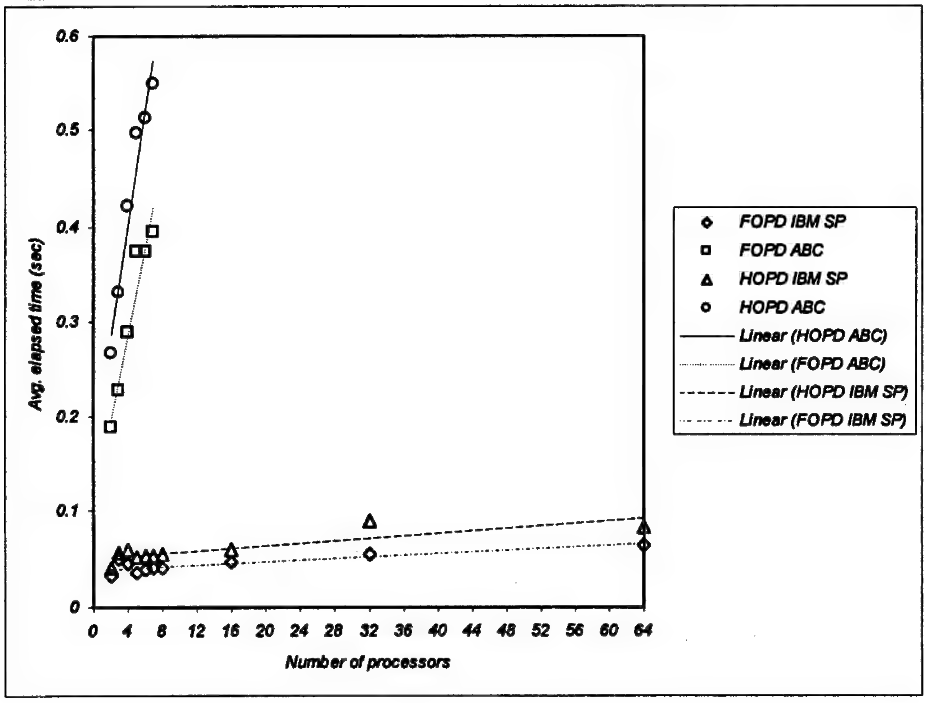


Figure 44 – Scatterplot showing the time spent in cornerturns (top) and source/sink communications operations (bottom): IBM SP vs. ABC.

As said before, the key factor for the IBM SP better scalability in our case study was the performance of the interprocess communication provided by the SP's high-performance switch. In order to show this effect, we plotted the times spent in the total exchange operations (cornerturns), and source/sink communication with the root processor. The cornerturn operations were the object of an interesting phenomena. In theoretical parallel computing, a common belief is that communication overhead increases with increasing machine sizes, but that was not totally true for FOPD and HOPD. As the scatterplot on Figure 44 shows, the cornerturn times actually *decrease* as the number of processors increase; this is attributed to the decreasing message size. Other observation that can be made is that the performance of collective operations on ABC degrades fastly with the increasing number of processors involved due to network contention [Culler98:154].

When source/sink communication is considered, the theory completely reflects practice: the more processors are added, higher is the time needed to enable communication between all of them and the root processor. However, the effects of this endpoint contention on elapsed times is much less intensive on the IBM SP when compared to ABC, as sketched on the bottom scatterplot in Figure 44.

It is not our primary objective to evaluate the IBM SP performance thoroughly. However, since we were able to get good scalability, we now show the results obtained for 64 processors, according to the real-time requirements of the benchmark. Table 26 shows this numerical figures.

first-order post-Doppler	Flops Count	Exec. Time (sec)	Benchmark Requirement (Gflops/sec)	% sustained
Video to I/Q conversion	57,016,320	0.034	1.77	94.72
Array calibration and Pulse comp.	67,633,152	0.020	2.10	100.00
Doppler processing	15,728,640	0.007	0.49	100.00
Weights computation	63,700,992	0.011	1.98	100.00
Weights application	3,932,160	0.001	0.12	100.00

high-order post-Doppler	Flops Count	Exec. Time (sec)	Benchmark Requirement (Gflops/sec)	% sustained
Video to I/Q conversion	78,397,440	0.059	2.43	54.92
Array calibration and Pulse comp.	92,995,584	0.037	2.88	86.92
Doppler processing	21,626,880	0.011	0.67	100.00
Weights computation	1,074,991,104	0.178	33.33	18.09
Weights application	16,220,160	0.004	0.30	100.00

Table 26 – Sustained performance of FOPD and HOPD on the IBM SP.

According to Table 26, the maximum sustained rate was 6 Gflops/sec during the weight computation stage of HOPD. Dividing this value evenly by the 64 processors results in 94.2 Mflops performed by each processor, which translates in 17.4% utilization for the POWER2 SC processor. Since this level of utilization was obtained with the original implementation of the benchmark, we believe that higher percentages can be achieved by using IBM SP's specialized libraries for linear algebra and signal processing. The reader can compare the results shown here with the ones obtained on the ABC for the modified versions of FOPD (on Table 17) and HOPD (on Table 20).

The result obtained for the weight computation phase on HOPD (18%) shows that we are still far from meeting the throughput requirements in terms of QR decomposition operations, and that the machine still can scale up relying on a sufficiently large problem size to get positive speedups.

5.7 Summary

In this chapter we reported and commented on the results of the experiments designed in chapter IV. We started with MPI, and saw that the interconnection network system was able to capitalize 86% of the peak capacity of 100Mbits/sec. Also, we made some considerations in regard to collective communications and synchronization.

The BLAS/LAPACK packages were considered next, and the ASCI Red BLAS demonstrated to be considerably more efficient on the use the memory hierarchy than the reference implementation from Netlib. Results of several tests were presented along with a practical example involving the QR decomposition of square and rectangular matrices.

In regard to the FFT, the implementations from the FFTPACK, downloaded and compiled as a library on ABC, showed competitive Mflops performance when compared to a reference implementation from Intel© (the Intel Math Kernel Library), designed specifically for Pentium Pro and Pentium II processors.

The sequential and parallel RT_STAP programs were analyzed in detail. The data collected showed that the modifications made to accommodate the BLAS and FFT routines provided positive results. The HOPD STAP implementation turned out to be more scalable than FOPD. Performance according to the benchmark specifications were reported, statistical analysis of the results was done to check the soundness of the experiments, and cross-checking of the timing measurements was performed via a parallel program visualization tool. Additional insights on RT_STAP implementation details were drawn from the use of this tool.

The performance of the RT_STAP benchmark was examined on two additional platforms: the AFIT NOW and the IBM SP. The comparisons made showed that the ABC scalability is severely compromised by its interconnection network and communication layer, and evidence of this influence was showed for cornerturns and scatter/reduction operations from/to the root processor. The next chapter summarizes our conclusions and provide suggestions for further research according to the observations made in this case study.

VI. Conclusions and Future Directions

6.1 Conclusions

In this chapter we address the questions in Chapter 1 posed by this thesis effort and indicate the significance and impact of the experimental outcomes. In general, the application of the case study observational method for research, and the selection of the MITRE RT_STAP Benchmark, are demonstrated to be effective choices. The first allowed us to develop a flexible experimental framework that enabled the collection of empirical data and the programming activity to occur concomitantly. In this sense, special effort was applied to experimentation in order to improve the statistical significance of this investigation. RT_STAP provided a realistic basis for the experiments, because of its design-to-specification approach to the STAP problem, in which a sequence of increasing complexity in terms of algorithm design corresponded to an associated increase in problem size. Also, the general approach used in the benchmark implementation reflected the contemporary programming practice in this field, namely a data parallel model for coding, the use of high level collective operations for message-passing, and a test-bench algorithmic construction (with data sources and data sinks) that is consistent with the embedded nature of signal processing systems.

Of course, a definitive answer in regard to the suitability of the ABC (or of any computing system) for parallel signal processing cannot be given, but we believe that our own experience with the daily use of a PC cluster, together with the results of the experimentation done in this work allow us make statements about the capabilities and performance of ABC with a reasonable probability of correctness.

A cost/performance analysis between ABC and the AFIT NOW clearly indicates the first as the winner. Based on data from Table 20 we can derive a value around **$30/Mflop/sec** for the ABC running RT_STAP: a maximum sustained rate of 902 Mflop/sec in the weight computation phase dividing $27,700 – the total cost for ABC hardware and software. The same evaluation done for the AFIT NOW produces a much higher ratio of approximately $650/Mflop/sec.

The ABC and the Linux operating system provided a stable and flexible environment for development and testing. However, the MPICH implementation running upon the TCP/IP protocol could not utilize the full bandwidth that can be delivered by the Fast Ethernet interconnection. Identical experiments done on the AFIT NOW showed that the TCP/IP was the bottleneck in this process.

The interconnection network imposed a severe negative impact on the scalability of ABC, and this process seemed to be accelerated by the fast speed of the Pentium CPU as the machine scaled up, especially for relatively less computationally intensive applications like first-order post-Doppler STAP. Collective communication and reduction operations were significantly affected and showed rapid degradation as the machine size increased. Comparisons made using the results from the IBM SP showed that the reduction in message size did not bring benefits for the cornerturn operations on ABC. This comparison indicates the current network latency needs improvement to allow the cluster to benefit from the reduced size messages as the system scales up. We also experienced some level of network contention during collective communication operations on ABC. Therefore, it is important that programs be designed to schedule communications appropriately in the current network topology.

Endpoint contention is another limitation present on the ABC interconnection network. In this case, a change in topology (such as to a fat tree) may provide relief to this problem by allowing an efficient implementation of software combining trees [Culler98:154]. However, tree structures possess the bisection problem: removing a single link near the root bisects the network. This effort may impose constraints to the current *bimodal* flexibility presented by ABC.

Although the floating point performance of the IBM SP POWER2 processor (at 135 MHz) is higher than in the Pentium II 400 MHz (17.6 against 12.4, according to SPECfp95 at http://www.spec.org), we did not find any experimental evidence that could confirm this assertion. ABC's single node performance was the best for all sequential and parallel implementations, and these results were also reflected in separate timings from all modules that comprise the RT_STAP benchmarks. All the execution times obtained with ABC were lower than the ones obtained by using the AFIT NOW, and sixteen IBM SP nodes were needed to meet the performance of six ABC processors running high-order post-Doppler STAP. An important lesson learned was the importance of always working with the best possible sequential code in order to get speedups that reflect realistic parallel performance.

The results of a case study often cannot be generalized. However, it seems reasonable to conclude that only coarse grain parallelism and properly scheduled collective communication operations should be explored on ABC with its current configuration, given the network contention and the overhead incurred to implement synchronization and point-to-point communication.

The ABC low-cost network provided a reliable environment for development of parallel code, and the positive results obtained by using the ASCI Red Pentium Pro BLAS and the FFTPACK packages are examples of effective software technology tracking that can enhance program performance without sacrificing portability. As the use of COTS hardware/software becomes mainstream, demonstrating easy-to-develop portable software for parallel computers is more important than creating complex optimized particular solutions. Overall, we believe that the excellent cost/performance ratio, the hardware/software flexibility, and the high computational capabilities showed by ABC yield a positive final evaluation, in spite of the limited scalability currently presented by the cluster for this particular case study.

This thesis research approaches the STAP technique from the computational point of view. In this regard, it is clear that research should be conducted to improve the performance of STAP at the stage responsible for generating the adaptive weights. The RMD algorithm [Reed74] provided very rapid convergence rate and great reduction in computation complexity, but the SMI technique using the QR decomposition is still computationally expensive. Therefore, it is desirable to improve the single node performance on the QR decomposition, in order to allow the application to benefit from the data parallel programming model. The overlapping of communication with computation can also be explored to reduce the effects of communication on the scalability of STAP parallel programs.

6.2 Future Directions

Guided by the objectives of our research, we conclude that benchmarks would be an effective way of conducting experiments because they generally allow repeatable and objective comparisons. However, given the extremely broad field of signal processing applications, the most important recommendation is to explore other examples of signal processing implementations and test them on ABC. Certainly, new and useful insights can be generated from this practice.

The possibility of applying modifications to the original benchmark code are by no means exhausted; new and wiser applications of the BLAS library routines are possible. Specifically, increasing the level of abstraction by fostering the application of level two and three of the BLAS can expose sufficient granularity to compilers, enabling the reuse of registers and reduction in memory access times.

The fact that STAP shows extreme real-time computational requirements suggests the use of considerable number of processors organized in a distributed memory fashion. However, current symmetric multiprocessor systems and distributed shared memory systems are an attractive alternative for future research on digital signal processing applications. In this environment, a programmer can better explore parallelization techniques available from compilers and multithreading programming concepts. These improvements can accelerate the parallel software development process while providing ease of use at the same time.

The Windows NT mode of the cluster can also be explored for RT_STAP. The Microsoft Visual C++© development environment and corresponding compiler have been receiving many improvements on the last years, as well as the implementations of the MPI standard for the Windows NT environment (MPI Pro© and PaTENT©).

Finally, research can be conducted in the direction of selecting (or developing) a more efficient communication layer to replace TCP/IP. This is a necessary step to take in order to enable applications to get the most from faster interconnections that ABC may incorporate in the future, as the cluster grows.

Appendix A

Parallel Computer Architectures

A.1 SIMD × MIMD

According to Flynn's taxonomy [Flynn72], multiprocessor systems are classified as SIMD or MIMD computers. In fact, the early multiprocessor systems were SIMD machines (Illiac IV, CM-2, MasPar-MP1) idealized in the early 60's and commercially available in the 70's [Culler98]. The key idea in those machines, as in the more recent SIMD machines, is to have a single instruction that operates on many data items at once, using many functional units, each of which has its own set of registers. They require less hardware than MIMD computers because they have only one global control unit. Furthermore, they require less instruction memory because only one copy of the program needs to be stored. That makes this architecture suitable for data parallel programs; that is, programs in which the same set of instructions are executed on a large data set. Another advantage of SIMD machines is that interprocessor data exchange requires less startup time because the communication of a word of data is like a register transfer made under the presence of a global clock [Kumar94]. Table A.1 below contains examples of SIMD machines.

Name	Max # of processors	Peak performance/proc	Max memory/proc	Comm BW/processor	Year	Topology
Alenia Quadrics QHX	128-2028	50 MFLOPS	16 MB	50 MB/sec	1994	3D mesh
CPP Gamma II Plus	4096	0.6 MFLOPS	516 MB	480 MB/sec	1995	2D mesh

Table A.1 - Two SIMD multiprocessors [Steen98].

Nowadays, these machines are used mostly for special-purpose applications, and the MIMD model is used for general-purpose multiprocessor architecture instead. This occurred mainly for the following reasons [Kumar94]:

- SIMD cannot take advantage of the high performance and cost advantages of microprocessor technology because designers of SIMD computers must build custom processors for the machine.
- The model is too inflexible. Different processing elements cannot execute different instructions in the same clock cycle. That is a problem for data parallel programs in which significant parts of the computation are contained in conditional statements.

Individual processors in a MIMD computer are more complex, because each processor has its own control unit. It may seem that the cost of each processor must be higher than the cost of a SIMD processor. However, it is possible to use general-purpose microprocessors as processing elements in MIMD computers. In contrast, the CPU used in SIMD has to be especially designed [Kumar94]. Hence, due to economy of scale, processors in MIMD computers may be both cheaper and more powerful than processors in SIMD computers.

Another issue is that of synchronization. SIMD computers offer automatic synchronization among processors after each instruction execution cycle. Hence, SIMD computers are better suited to parallel programs that require frequent synchronization.

We can classify SIMD machines in terms of the physical organization of memory. When the available memory is distributed among the processors these machines are termed as Distributed-memory SIMD machines (DM-SIMD), or processor-array machines [Hockney88]. Most DM-SIMD systems have the possibility to handle I/O independently from the front/end processors. This is not only favorable because the communication between the front-end and back-end systems is avoided. The I/O devices for the processor-array system are generally much more efficient in providing the necessary data directly to the memory of the processor array. Especially for very data-intensive applications like radar and image processing such I/O systems are very important [Steen98]. SIMD machines with a physically unique shared-memory (SM-SIMD) are equivalent to vector-processors, to be described next.

A.2 Vector Supercomputers

While vector-processors are a simple, special case of SIMD machines, they are usually considered as a different class because vector registers and vector instructions appear in the processors of many parallel MIMD systems as well [Censor97]. Also, the development of algorithms or software for a vector computer poses different problems than the design of algorithms for a SIMD system.

High-speed, pipelined processors are particularly useful for large scientific and engineering applications. A high-speed pipelined processor will usually use a cache to avoid forcing memory reference instructions to have very long latency. Unfortunately, big, long-running, scientific programs often have very large active data sets that are sometimes accessed with low locality, yielding poor performance from the memory hierarchy.

Another problem with scientific parallel applications that require manipulation of vast amounts of data is the I/O bottleneck [Pasquale94]. The reason for that is the broadening disparity in the performance of I/O devices and the performance of processors/communication links on parallel and distributed-processing platforms. The first I/O characterization efforts for scientific applications were done in vector-processors [Pasquale94], and the behavior was found to be regular and predictable. However, more recent equivalent studies in MIMD systems diagnosed greater variability and irregularity, and current research exists with the objective of develop standard application programming interfaces (API) to provide grater performance for parallel I/O [Smirni97].

Vector multiprocessors rely on pipelined functional units that typically operate on several vector elements per clock cycle, by using high-level operations that work on vectors – linear arrays of numbers. Programs that have very large active data sets that are accessed with low locality can benefit from vector processing due to some particular characteristics of these machines, such as [Patterson98]:

- Vector instructions that access memory have a known access pattern. If the vector's elements are all adjacent, the fetching the vector from a set of heavily interleaved memory banks works very well. The high latency of initiating a main memory access versus accessing a cache is amortized, because a single access is initiated for the entire vector rather than to a single word. Thus, the cost of the latency to main memory is seen only once for the entire vector, rather than once for each word of the vector.
- The computation of each result is independent of the computation of the previous result, allowing a very deep pipeline without generating data hazards. Because an

entire loop may be replaced by a vector instruction whose behavior is predetermined, control hazards from the loop branch are nonexistent.

These advantages mean that vector multiprocessors do not need to rely excessively on high hit rates in cache to have high performance. In fact, the majority of vector-processors today do not employ a cache anymore [Steen98]. They tend to rely on low-latency main memory, often made of SRAM, and have as many as 1024 memory banks to get high memory interleaved bandwidth. Table A.2 shows two examples.

Name	Vector registers	Elements per vector register	Elements computed per clock cycle	Number of FU	Processor clock rate	Max # of processors	Max memory size/system
Cray T90	8	128	2	8	455MHz	32	8,192 MB
Fujitsu VPP300	8-256	64-2048	8	4	140Mhz	16	32,768 MB

Table A.2 - Two vector computers for sale in 1997 [Steen98].

A.3 MIMD Multiprocessors

Given how the state-of-the-art in parallel architectures has advanced, we need to take another look at how to classify the existing variety on the field. The traditional taxonomy SIMD/MIMD is of little help these days. One cannot just look at the hardware structure, since common elements are employed in many different ways. Instead, we ought to focus our attention on the architectural distinctions that make a difference to the software that is to run on the machine [Culler98].

Existing MIMD machines fall into two classes, depending on the number of processors involved, which in turn dictate a memory organization and interconnect strategy. We refer to the machines by their memory organization, because what constitutes a small or large number of processors is likely to change over time [Hennessy96].

The first group, which we call centralized shared-memory architectures, have at most a few dozen processors in the mid 90's. For multiprocessors with small processor counts, it is possible for the processors to share a single centralized memory and to interconnect the processors and memory by a single bus. With large caches, the bus and the single memory can satisfy the memory demands of a small number of processors. Advances in memory bus technology, including faster electrical signaling, wider data paths, pipelined protocols, and multiple paths provided greater bandwidth, and the acceptance of more processors [Culler98]. Figure A.1 gives us an idea of the bandwidth of the shared memory bus in commercial SMPs.

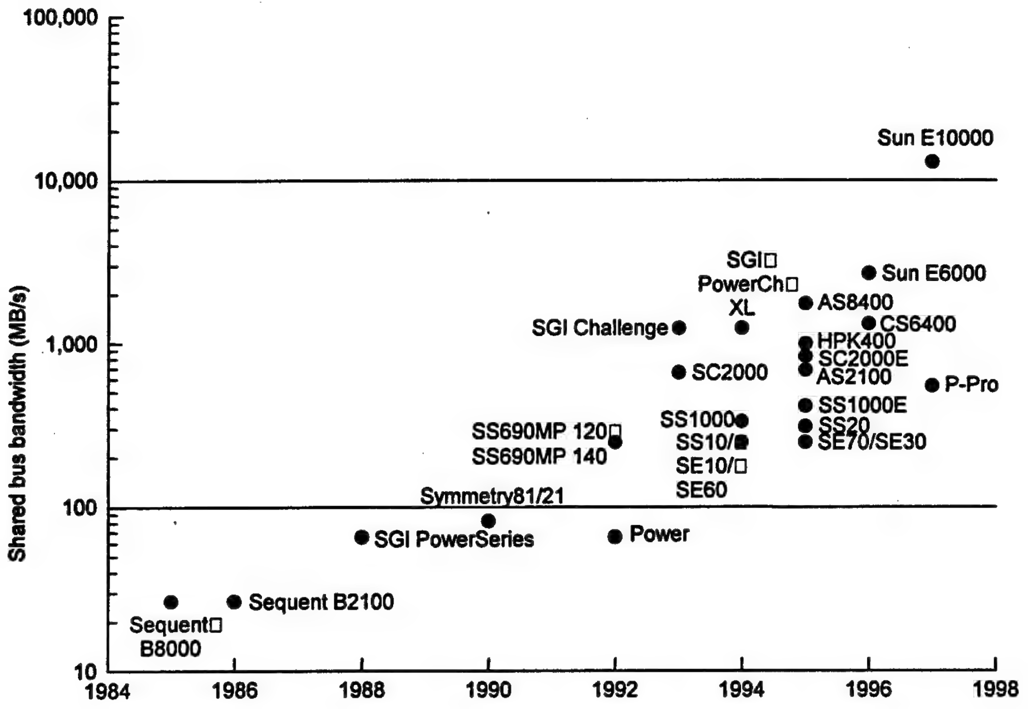


Figure A.1 – Memory bandwidth in commercial SMP's [Culler98].

Because there is a single main memory that has a uniform access time from each processor, these machines are sometimes called Uniform Memory Access machines (UMA) or Symmetric Multiprocessors (SMP). Table A.3 presents information on some SMPs commercially available.

The term UMA comes from taxonomy of parallel computers that used to be popular primarily with those who designed operating systems [Pfister98]. In this classification, we also have:

- NUMA – non-uniform memory access, where every processor has access to all of memory using normal loads and stores. However, there is a noticeable different delay, depending on what parts of memory are accessed; hence "non-uniform".

- NORMA – non-remote memory access, that is, processors cannot access other processors' memories by normal loads and stores, but instead must communicate by other means. For practical purposes, this is another term for message-passing systems.
- COMA – cache-only memory access. That is much like NUMA, except for the fact that all movement of data is done by hardware, not software, and there is no main memory, only multiple levels of caches with ever-larger line sizes reaching the point where the cache lines are like distributed virtual memory (page-sized) [Rothnie92].

Name	Max # proc.	Processor	Clock	Max memory/system	Comm. BW/system
Compaq ProLiant 5000	4	Pentium Pro	200 MHz	2,048 MB	540 MB/sec
Digital AlphaServer 8400	12	Alpha 21164	440 MHz	28,672 MB	2150 MB/sec
HP 9000 K460	4	PA-8000	180 MHz	4,096 MB	960 MB/sec
SGI Power Challenge	36	MIPS R10000	195 MHz	16,384 MB	1200 MB/sec
Sun Enterprise 6000	30	UltraSPARC 1	167 MHz	30,720 MB	2600 MB/sec
IBM RS/6000 R40	8	PowerPC 604	112 MHz	2,048 MB	1800 MB/sec

Table A.3 - Some SMPs connected by a single bus, for sale in 1997 [Patterson98].

The second group consists of machines with physically distributed memory called distributed memory architectures. To support larger processor counts, memory must be distributed among the processors rather than centralized; otherwise the memory system would not be able to support the bandwidth demands of a larger number of processors[Hennessy96]. Distributing memory among the nodes has two major benefits:

- It is a cost effective way to scale memory bandwidth, if most of the accesses are to the local memory in the node (locality of reference).
- It reduces the latency for accesses to the local memory.

The key disadvantage for a distributed memory architecture is that communicating data between processors becomes more complex and has higher latency because the processors no longer share a single centralized memory. The ratio of remote to local access times can be as small as 2 to 1 for the SGI Origin 2000 or as high as 8 to 1 for the sequent STiNG [Sound98]. As we will see next, the use of distributed memory leads to two different paradigms for interprocessor communication.

On the other hand, centralized shared-memory architectures that employ caches have to deal with the cache coherence problem. Here, the system must provide a coherent, uniform view of the memory to all processors, despite the presence of local, updateable, private cache storage. In order to accomplish this objective, several cache-coherent protocols have been developed, such as the Scalable Coherent Interface – SCI [Gustafson92], and the Stanford DASH protocol [Lenoski92]. CC-NUMA is the term used to refer to the implementation of a cache coherent memory on a NUMA machine.

A.4 Memory Addressing and Communication Mechanisms

As stated earlier, any large-scale multiprocessor must use multiple memories that are physically distributed with the processors. In addition, there are two architectural approaches for addressing the memory available:

- The physically separate memories can be addressed as one logically shared address space, meaning that a memory reference can be made to any memory location by any processor, assuming the correct access rights. Machines of this kind are called

Distributed Shared-Memory machines (DSM) or Non-Uniform Memory Access machines (NUMA), since the memory access time depends on the location of a data word in memory. Here, the term shared-memory refers to the fact that the address space is shared; that is, the same physical address on two processors refers to the same location in memory. Table A.4 below lists some characteristics of DSM multiprocessors commercially available.

Name	Max # of processors	Processor	Processor clock rate	Max mem/sys	Communication BW/link	Node	Topology
Cray T3E	2048	Alpha 21164	450Mhz	524,288 MB	1200 MB/sec	4-way SMP	3d-torus
HP/Convex Exemplar	64	PA-8000	180MHz	65,536 MB	980 MB/sec	2-way SMP	8-way crossbar + ring
Sequent NUMA-Q	32	Pentium Pro	200MHz	131,072 MB	1024 MB/sec	4-way SMP	Ring
SGI Origin2000	128	MIPS R10000	195MHz	131,072 MB	800 MB/sec	2-way SMP	6-cube

Table A.4 - Some DSM multiprocessors for sale in 1997 [Patterson98].

- The address space can consist of multiple private address spaces that are logically disjoint and cannot be addressed by a remote processor. In such machines, the same physical address on two different processors refers to two different locations in two different memories. Each processor-memory module is essentially a separate computer; therefore these machines are called multicomputers [Hennessy96].

With each of these approaches to memory addressing there is an associated communication mechanism. For a machine with a shared address space, that same address space can be used to communicate data implicitly via load-store operations; hence the name shared-memory for such machines. For a machine with multiple address spaces, communication of data is done by explicit passing messages among the processors. Therefore, these machines are called message-passing machines [Hennessy96].

A.4.1 Message Passing Interface (MPI)

At the Supercomputing 1992 conference, a committee, later known as the MPI Forum, was formed to develop a message-passing standard [MPI]. Prior to the development of the Message Passing Interface (MPI) in 1994, each parallel computer vendor developed a custom communication protocol.

Since its introduction, the main goal of the MPI design was portability, and it is achieved by providing a public domain, platform-independent standard of message-passing library. MPI specifies this library in a language-independent form, and provides Fortran and C language bindings. The specification does not contain any feature that is specific to any particular vendor, operating system, or hardware [Hwang98].

There are several implementations of MPI. Examples include the CHIMP developed at the Edinburgh University, and the MPICH, which is the most popular implementation, developed jointly by Argonne National Laboratory and Mississippi State University. MPICH is a widely used, portable implementation of MPI that runs on PCs, NOWs, SMPs and MPPs.

Additionally, the use of MPI collective operations can significantly increase programming productivity; specifically, portable versions of common signal processing communication kernels such as the distributed matrix cornerturn [Games98] can be constructed and used instead of proprietary programming environments.

A.5 Massively Parallel Processors and Clusters

In order to take advantage of higher parallelism available in applications such as scientific computing, engineering simulation, signal processing, and data warehousing, we need to use even higher scalability computer platforms by exploiting the distributed memory architectures described before.

Recently, we have been able to see a clear convergence for scalable machines towards a generic parallel machine organization [Culler98]. An MPP is an example of this trend, and comprises a collection of essentially complete computers, each with one or more processors and memory, connected through a scalable communication network with high bandwidth and low-latency.

The term massively parallel processor generally refers to a very large-scale computer system having the following characteristics [Hwang98]:

- It uses commodity microprocessors in its processing nodes.
- Each node holds a copy of an optimized micro-kernel, not a full OS.
- It uses a custom interconnect with high communication bandwidth and low latency.
- It can be scaled up to hundreds or even thousands of processors.
- The communication mechanism is by message passing.

Name	# of processors	Processor	Memory size/ system	Comm BW/link	Node	Topology	Peak performance TFLOPS	Year
ASCI Red	9216	200 Mhz Pentium Pro	580,608 MB	800 MB/sec	2-way SMP	Two 2D grids	1.8	1996
Avalon A12	1680	Alpha 21164	1.7 TB	10 GB/sec	1-way	Multistage	1.3	1996

Table A.5 - Characteristics of two MPPs [Steen98].

As can be seen from Table A.5 above, MPPs generally employ commodity-off-the-shelf microprocessors on its nodes. Although this practice has advantages (such as reduction in cost and incorporation of new technologies) it also require some hardware adjustments because microprocessors are not designed to operate in large systems [Hwang98]. Most of the problems are related to accessing and addressing available memory.

As an alternative for building large-scale machines we have the clustering of workstations, SMPs, and PCs as a trend in developing scalable parallel computers. The main idea is to design a machine using all off-the-shelf components, which promises the lowest cost and rapid technology tracking [Anderson95]. The leverage in this approach lies in the use of commodity technology everywhere: in the processors, in the interconnect (Ethernet, FDDI, Fiber-Channel, and ATM switch), and in the software (standard operating systems and programming languages). Examples include the IBM SP2, and the Berkeley NOW [Culler97]. Table A.6 has more examples.

Name	Max # processors	Processor	Processor Clock rate	Max memory size/system	Comm BW/link	Node	Max # of nodes
HP 9000 EPS21	64	PA-8000	180 MHz	65,536 MB	532 MB/sec	4-way SMP	16
IBM RS/6000	16	PowerPC 604	112 MHz	4,096 MB	12 MB/sec	8-way SMP	2
IBM RS/6000 SP2	512	Power2 SC	135 MHz	1,048,576 MB	150 MB/sec	16-way node	32
Tandem Himalaya S70000	4096	MIPS R10000	195 MHz	1,048,576 MB	40 MB/sec	16-way SMP	256

Table A.6 - Some clusters commercially available in 1997 [Patterson98].

Several issues must be considered in developing and using a cluster. Although much work has been done, they are still active research and development areas. Four important issues are introduced below [Anderson95][Pfister98][Hwang98].

- Availability support – Clusters can provide cost effective high availability with lots of redundancy in processors, memories, disks, I/O devices, etc. However, to realize this potential, specially designed software has to be developed; typically a set of daemons running on the nodes and exchanging messages between themselves.
- Single system image – A set of workstations or PCs connected by an Ethernet is not necessarily a cluster, for a cluster is a single system. By clustering, say, 100 workstations through a Single System Image technique (SSI) we should theoretically get a 'megastation' 100 times more powerful. This is an appealing goal, but very difficult to achieve.

- Job and resource management – Clusters try to achieve high system utilization, out of traditional workstations or PC nodes that are normally not highly utilized. However, especial management is needed to provide batching, load-balancing, parallel processing, and other functionality.
- Efficient Communication – It is more challenging to develop an efficient communication subsystem for a cluster than for an MPP. Due to higher node complexity, cluster nodes cannot be packaged as compactly as MPP nodes. Long wires implies larger interconnect network latency and reliability problems that demand secure communication protocols, which increase overhead. Clusters often use commodity networks with standard communication protocols such as TCP/IP that have high overhead. Currently there is no accepted standard for low-level communication protocol.

As mentioned earlier, a number of libraries are already available to allow parallel computing on network of workstations. Libraries like MPI and PVM are built on top of the TCP/IP protocol. The performance offered by TCP/IP and consequently delivered to the libraries built on top of it, is one or two orders of magnitude slower than that available on MPPs. Using faster medium than Ethernet does not bring much improvement [Liu94]

To overcome this problem, many research efforts have been conducted, like in [Anderson95][Pakin97] in order to address critical issues such as division of overhead between host and network coprocessor, I/O management for the bus and the buffers, and the mapping of the network device interface directly into the user address space to allow faster data access via load/store operations.

A.5.1 Pile-of-PCs

A similar architectural concept of NOWs is the "Pile of Personal Computers", or cluster of PCs [Ridge97]. Much of the same research performed on clusters of traditional workstations are also being performed on clusters of personal computers, such as the Princeton SHRIMP, which developed a network-oriented VM scheme for PCs using a custom-built network [Hwang98]. Others, such as the Real World Computing Partnership's COMPaS, use COTS networks and PCs to create hybrid SMP/DSM systems using a commercial UNIX implementation [Tanaka98].

Also, there are products, in development or already developed, which provide distributed computing capability on Windows NT Servers/workstations. One of these is Microsoft's own NT Clusters software, formerly known as Wolfpack, which is still in development. The initial release primarily focuses on fault tolerance and not on distributed computing [Pfister98]. A second product, developed at Mississippi State University, is MPICH/NT. A commercial version of the software, MPI/Pro, is available from MPI Software Technology. This software has been acquired by AFIT and is currently loaded on the ABC, to allow parallel computing under the Windows NT OS. The software allows both shared-memory (for SMP clusters only) and message-passing configurations.

The University of Coimbra, in Portugal developed another product, WMPI. This product is a port of the P4 device to Windows NT. It targets TCP/IP networks and does not support threads at this time [Hebert98]. A fourth product is the High Performance Virtual Machine (HPVM) developed by the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign (UIUC)

[UIUC98]. Underlying this software is a fast messaging version of MPI, MPI-FM, also developed at UIUC. The HPVM software is also currently loaded on the ABC system.

A.5.2 Beowulf

The Beowulf-class computer, so named after the prototypical system, is built exclusively from commercial-off-the-shelf (COT) hardware to take full advantage of the economy of scale that commodity computers and commodity networks offer. The operating system is an open-source, free-license system such as Linux or FreeBSD, which significantly reduces the expense of the system, especially when scaled to hundreds of processors, and allows optimization of the operating system to the particular architecture [Ridge97].

NASA started the Beowulf project in 1994 with the mandate of developing a "Gigaflops Scientific Workstation" for Earth and Space Sciences (ESS) applications. Following NASA's philosophy, the Beowulf team built their prototype using sixteen Intel 80486DX4-based personal computers with 10BaseT Ethernet and 10Base2 channel-bonded Ethernet and free-license software (Linux) that allowed optimization of the operating system for the architecture and application [Sterling95]. By 1996, a Beowulf-class system constructed from sixteen Intel Pentium Pro machines networked by dual 100BaseT switched Ethernet was able to sustain 1.25Gflops for less than $50,000 [Ridge97].

Since the original was announced, several Beowulf-class systems have been constructed throughout the world by government research laboratories, academic institutions, and commercial vendors, taking advantage of the very low price afforded by the economies of scale available from commodity PCs. These systems have been based on not only the Intel x86 architecture, but also IBM PowerPCs, Sun SPARCs, and DEC Alphas, and have employed interconnection network topologies including shared-bus, ring, torus, hypercube, and shallow fat-tree built from Ethernet, Fast Ethernet, or Myrinet [Ridge97]. Table A.7 shows some Beowulf clusters currently set up.

Institution	Number of processors	Type of processor on each node	Operating system
Los Alamos Center for Nonlinear Studies	140	DEC Alpha 21164A – 533 MHz	Linux
University of Wisconsin	48	DEC Alpha 21164 – 300 MHz	Linux
Goddard Space Flight Center	64	Pentium Pro – 200 MHz	Linux
California Institute of Technology	114	Pentium Pro – 200 MHz	Linux
Air Force Institute of Technology	12	Pentium/Pentium II - 200 to 450 MHz	Linux / Windows NT
Sandia National Laboratories	14	Quad-Pentium Pro – 200 MHz (SMP)	Linux / Windows NT

Table A.7 – Some Beowulf clusters available.

Appendix B

Platforms for Research

B.1 AFIT Bimodal Cluster - ABC

For the purposes of this research, the AFIT cluster of PCs is a *dedicated* shared-nothing parallel machine consisting of one Dell 450 MHz Pentium II processor, six Dell 400 MHz Pentium II processors, one Dell 200 MHz Pentium processor, and four Gateway 333 MHz Pentium II processors connected via a 100 Mb/sec full duplex 24-port switched Fast Ethernet – the average delay through the switch is 11 microseconds. The switch has an aggregate internal bandwidth of 6.3 Gbit/s and an aggregate network bandwidth of 800 Mbit/s. Each processor can be booted either running Windows NT 4.0 or Linux 2.0.33 operating systems. Parallel communication is handled through MPI/Pro 1.2.3 or Patent MPI 4.0 for Windows NT, and MPICH version 1.1 for Linux applications.

Three of the four Gateways have 128 Mb 15 nsec SDRAM, and each of the Dell processors has 128 MB of 10 nsec SDRAM. The fourth Gateway has 256 Mb 15 nsec SDRAM. The Pentium 200 MHz has 32 Mb of main memory. The Pentium II processor Level 1 cache consists of a 4-way set associative 16 KB instruction cache and 16 KB nonblocking 2-way set associative dual ported data cache. The Level 2 cache is 512 KB nonblocking, squashing, unified 4-way set associative physically addressed L2 cache capable of handling four outstanding misses and has a twelve entry load queue. The L2 cache is clocked at half the speed of the processor.

Under the NT configuration, each Gateway processor has one 8 GB EIDE hard drive at its disposal; whereas, the Dell computers have one 8.4 GB SCSI hard drive. When the system is Linux, each processor (Gateway or DELL) has one 5.6 GB EIDE hard drive available, except one that has a 540 MB EIDE hard drive.

Finally, the I/O bus on the Gateways and on the Pentium 200 MHz operates at 66 MHz whereas the Dell's I/O bus is clocked at 100 MHz.

B.2 The AFIT NOW

It consists of five Sun Ultra Sparc© workstations model 170 (170 MHz processor) and one of model 200 (200 MHz processor), connected via the high-speed Myrinet© switch. The processors are four-way superscalar of version 9, with two integer ALU units and two pipelined FP ALUs. There is a 16 Kbyte direct-mapped data cache and a 16 Kbyte 2-way set associative instruction cache, both on-chip. The level-2 cache has 512Kbytes. Each workstation has 128 Mbytes of RAM and two 1Gbyte local hard disk drive.

The Myrinet network includes an 8-by-8 crossbar switch, and each link provides 1.28 Gbits/sec in each direction. The Network Interface Card (NIC) is composed of a CISC processor that operates at 25 MHz and achieves approximately 5 MIPS. The NIC has 128Kbytes of SRAM. Its memory is mapped into the DMA address space of the Sparc processor. The protocol used in the messaging layer is TCP/IP. The MPI implementation used for communication is MPICH 1.0.1.

B.3 The ASC MSRC IBM-SP

The IBM SP system is a scalable distributed memory multicomputer based on the IBM RS/6000 Power2 SC 4-issue superscalar processor [IBM98] operating at 135 MHz, and capable of deliver 540 Mflops peak. From its 256 processors, 233 are available for computation, each one with 1 Gbyte of main memory. There is a 4-way set associative, 128 Kbytes data cache and a 32 Kbyte instruction cache. The NIC includes a Power PC 601 processor that performs DMA. There are 244 Gbytes of memory in total, and 2 High-Performance Parallel Interfaces (HiPPi). The total disk capacity is 2.7 Tbytes, with 420 Gbytes available in the working space. The interconnection network is a multistage Omega network, with theoretical bandwidth of 40 MB/sec per link in each direction. The operating system is the AIX 4.1.

Appendix C

RT_STAP Algorithm Description and Complexity

The purpose of this appendix is to describe the RT_STAP implementation, according to the level of observation for this research, in order to (a) provide a basic understanding of the application structure, complexity and functionality, and (b) guide the subsequent design of the experiments and selection of the performance metrics. The appendix is organized upon the modules that build the benchmark: preprocessing, nonadaptive Doppler filtering, subsequent adaptive processing, and the global communication steps represented by the distributed STAP cornerturns. Much of the contents of this appendix comes directly from the benchmark specifications in [Cain97].

The RT_STAP parallel implementations are designed around the data-parallel programming model [Culler98:26]. Here, several processors perform the operations corresponding to a given phase asynchronously on separate elements of the input data set and then exchange information globally before continuing to the next phase. The global exchange / reorganization phase (cornerturns) is accomplished through the use of messages, employing the *MPI-Alltoallv* collective operation.

The process which rank is 0 in the *MPI_Comm_World* communicator [Pacheco97] is responsible for reading the input data file into local memory, partition it in according to the number of processors (up to 64), and send each of these chunks to the other processors in the communicator using *MPI_Scatterv*. The rank 0 processor is also responsible for reading the parameters file and broadcast the information to the other processors using *MPI_Bcast*. By the end of processing, the processors with rank ≠ 0 send the results back to processor 0 , a gathering operation accomplished by *MPI_Gatherv*. The output range-Doppler data matrix information is then validated and written in an output file. All the processing steps are executed sequentially on each node without communication. Figure C.1 provides a graphical representation of this implementation.

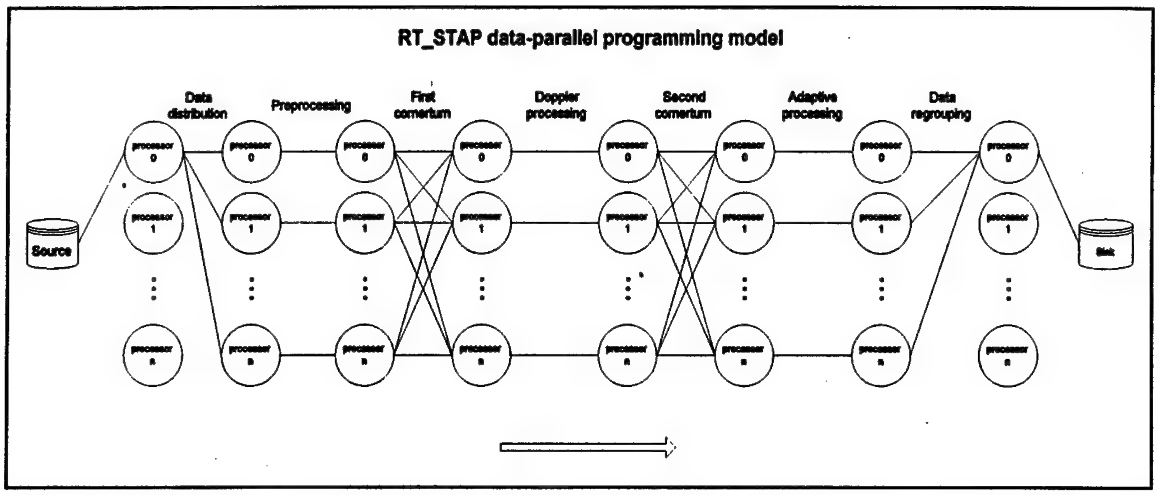


Figure C.1 – Graphical representation of RT_STAP parallel implementation.

C.1 Preprocessing

Before applying space-time adaptive processing algorithms to data samples measured by an airborne antenna array, a significant amount of preprocessing must be performed. There are three major functions: video-to-I/Q conversion, array calibration, and pulse compression. These functions are applied to the A/D data samples independently across the L channels. Complex data samples are passed between each component of the preprocessing with the data at the output of the pulse compression functional block forming the input to the STAP. Figure C.2 shows a block diagram of the STAP preprocessing.

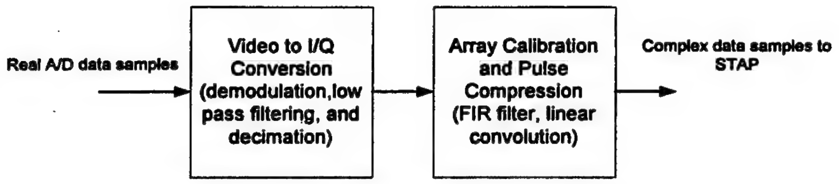

Real A/D data samples
Video to I/Q Conversion (demodulation,low pass filtering, and decimation)
Array Calibration and Pulse Compression (FIR filter, linear convolution)
Complex data samples to STAP

Figure C.2 – Preprocessing for RT_STAP.

To support the STAP algorithm, a data cube corresponding to the L channels, P pulse repetition intervals (PRIs), and N time samples per PRI, must be processed (see Figure C.3). This data cube corresponds to a single CPI of the radar system. On input, these data samples are real values. We define $x(l,p,n)$ to be a real data sample corresponding to the n^{th} time sample from the p^{th} PRI of the l^{th} channel of the CPI.

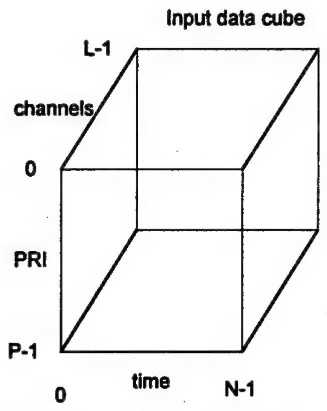


Figure C.3 – Input data cube for a single CPI

Demodulation to baseband is achieved by multiplying the data by demodulation coefficients (i.e., complex sinusoid) that translate the signal to baseband. If we let f_{IF} denote the center frequency of real data samples, $f_{\mathrm{A\text{-}D}}$ be the sampling frequency of the data, and $h_d(p,n)$ be the complex demodulation coefficients, then the output after frequency translation is:

$$x_0(l,p,n) = h_d(p,n)x(l,p,n),$$

where $n = 0,\ldots,N-1$, $p = 0,\ldots,P-1$, $l = 0,\ldots,L-1$, and $h_d(p,n)$ is given by:

$$h_d(p,n) = 2\exp\{-j2\pi f_{IF}(n+pN)/f_{A-D}\}.$$

Implementation of the demodulation to baseband function requires $2.N$ floating-point operations per channel and PRI, resulting in a total of $L.P.2N$ floating-point operations.

Anti-aliasing is then accomplished with a finite impulse response (FIR) lowpass filter. The length of the filter is defined to be K_a and the real-valued filter coefficients are denoted by $h_a(k)$ for $k = 0,\ldots,K_a - 1$. The output of the filter corresponds to the discrete linear convolution of the real filter coefficients with the complex data:

$$x_1(l,p,n) = \sum_{k_a=0}^{K_a-1} h_a(k_a)x_0(l,p,n-k_a),$$

for $n = 0,\ldots,N-1$, $p = 0,\ldots,P-1$, and $l = 0,\ldots,L-1$.

After applying the lowpass filter, the sample rate conversion function decimates the data to achieve the desired data rate. The data is to be decimated by an integer value, D, defined to be the ratio of the sampling rate of the data on input to the preprocessor and the desired sampling rate after conversion. The final output of the conversion process is:

$$x_2(l,p,n_D) = x_1(l,p,n_D \cdot D),$$

where $n_D = 0,\ldots,N_D - 1$, $p = 0,\ldots,P-1$, $l = 0,\ldots,L-1$, and $N_D = \lfloor N/D \rfloor$.

Lowpass filtering is implemented using discrete linear convolution. FIR filtering and decimation function requires $3 \cdot K_a \cdot N_D$ floating-point operations per channel and PRI, resulting in a total of $L \cdot P \cdot (3 \cdot K_a \cdot N_D)$ floating point operations.

A combined implementation of calibration and pulse compression was adopted for this implementation, which requires the computation of the following expression:

$$x_4(l,p,n_D) = x_2(l,p,n_D) * h_c(l,n_D) * h_p(n_D) = x_2(l,p,n_D) * h_{cp}(l,n_D),$$

where "*" denotes discrete linear convolution over the index n_D, h_c and h_p are FIR filter coefficients used in calibration and pulse compression, respectively. It is assumed that the convolution of the filter coefficients is performed off-line to produce a combined filter response of $h_{cp}(l,k)$, corresponding to a sequence of length $K_{cp} = K_c + K_p - 1$, where K_c and K_p are the FIR filter lengths used in array calibration and pulse compression, respectively. The combined coefficients are computed using:

$$h_{cp}(l,n_D) = \sum_{k_c=0}^{K_c-1} h_c(l,k_c)\, h_p(n_D - k_c),$$

for $n_D = 0,\ldots,K_{cp} - 1$ and $l = 0,\ldots,L-1$. Note that $h_c(l,k) = 0$ for $k < 0$ or $k \geq K_c$ and $h_p(k) = 0$ for $k < 0$ or $k \geq K_p$. The output of the preprocessing corresponds to the discrete linear convolution of the combined filter coefficients and the set of length N_D sequences of data samples, $x_2(l,p,n_D)$, for $p = 0,\ldots,P-1$ and $l = 0,\ldots,L-1$.

The overlap-save [Proakis96:430] method is used to implement the discrete circular convolution of segments of the complex data and the filter coefficients, followed by a selection of the part of the circular convolution corresponding to the linear convolution of the two sequences. The complex data $x_2(l,p,n_D)$ is augmented with

$K_{cp}-1$ leading zeros. The augmented data sequence is then divided into B overlapping segments of length $\tilde{R}+K_{cp}-1$, where $\tilde{R}$ and B are related through the expression $B=\lceil N_D/\tilde{R}\rceil$ and $\tilde{R}$ is the length of the discrete linear convolution. Each segment is overlapped by $K_{cp}-1$ samples. FFTs are applied to both the data block and the sequence of filter coefficients with both sequences zero padded so that the length of the FFT is a power of two. The length of the FFT, R, is set to $\tilde{R}+K_{cp}-1$. As a result of this choice of $\tilde{R}$, only the last data block is need to be zero padded.

Once the FFTs are computed, the transformed sequences are multiplied and an inverse FFT is applied to the result to obtain the time-domain representation of the circular convolution. Samples 1 through $K_{cp}-1$ are discarded and the remaining samples from the B data segments are assembled to form the final output of the preprocessing.

Implementation of the FFTs and inverse FFTs used to compute the convolution requires $5\cdot R\cdot\log_2 R$ floating-point operations per FFT or inverse FFT [Golub96:190]. Multiplication of the sequences in the frequency domain requires $6\cdot R$ floating-point operations per data block. The total operation count for calibration and pulse compression processing of the entire data cube is $L\cdot P\cdot B\cdot(10\cdot R\cdot\log_2 R+6\cdot R)$, where $R=K_c+K_p+\tilde{R}-2$ and $B=\lceil N_D/\tilde{R}\rceil$. The number of data blocks, B, and the length of the FFT, R, are selected to minimize the computational complexity of the implementation, and their values vary as a function of the number of time samples per PRI. Table C.1 summarizes the computation counts for the preprocessing phase of RT_STAP,

Function	Operation Count
Video-to-I/Q Conversion	$L \cdot P \cdot (2 \cdot N + 3 \cdot K_a \cdot N_D)$
Calibration and Pulse Compression	$L \cdot P \cdot B \cdot (10 \cdot R \cdot \log_2 R + 6 \cdot R)$; $R = K_c + K_p + \tilde{R} - 2;\;\; B = \lceil N_D / \tilde{R} \rceil$

Table C.1 – Computation counts for preprocessing.

where the parameters in the expressions are defined as:

L:	Number of channels
P:	Number of PRIs per CPI
N:	Number of samples per PRI before decimation
D:	Decimation factor
N_D:	Number of samples per PRI after decimation; $N_D = \lfloor N/D \rfloor$
K_a:	FIR filter length used for anti-aliasing in video-to-I/Q conversion
K_c:	FIR filter length used in array calibration
K_p:	FIR filter length used in pulse compression
R:	FFT size (power of 2) used by the overlap-save fast convolution method
$\tilde{R}$:	Linear convolution length in array calibration and pulse compression
B:	Number of blocks in the overlap-save convolution method. $B = \lceil N_D / \tilde{R} \rceil$.

C.2 Doppler Processing

The first component of all three post-Doppler adaptive algorithms presented in the RT_STAP benchmark is Doppler processing. It is implemented by applying a discrete Fourier transform (DFT) of length K across P pulses of the preprocessed data for a given range cell and channel, where K represents the number of Doppler cells to be processed. A pre-computed window function is applied to the data to reduce spectral leakage. The K complex data samples after Doppler processing can be written as:

$$x_5(l,k,r) = \sum_{p=0}^{P-1} d(p) x_4(l,p,r) e^{j(2\pi/K)pk},$$

for $r = 0,\ldots,N_D - 1$, $k = 0,\ldots,K-1$, and $l = 0,\ldots,L-1$. In the above expression, the quantity $d(\cdot)$ represents the real-valued window function applied to the data samples (Rectangular, Hanning, Hamming, or Blackman), where $\vec{d}^{\mathrm{T}}\vec{d}$ equals unity.

The DFT is implemented using a FFT algorithm. The data samples are zero padded, if necessary, so that the length of the FFT, denoted by K, is a power of two. A window function having length P is applied to the data before the transformation is performed. Application of the real-valued window function across all pulses of a given range cell and channel requires $2 \cdot P$ floating-point operations. Therefore, a total of $5 \cdot K \cdot \log_2 K + 2 \cdot P$ operations is needed to implement Doppler processing for a given range cell and channel. The number of required Doppler processing functional blocks depends on the number of range cells, channels, and type of adaptive processing algorithm. The total number of computations required to implement Doppler processing is $L \cdot N_D \cdot (5 \cdot K \cdot \log_2 K + 2 \cdot P)$.

C.3 Adaptive Processing

The fully adaptive approach considered in the last chapter is impractical for reasons of computational complexity and sample support required for weight training. Partially adaptive techniques represent a way of reducing the dimensionality of the problem, and they were organized in a taxonomy devised by Ward [Ward94] and described in section 2.6.2.

The input data originates from a sampling in time from the pulses of the CPI and in space at the locations of the antenna array elements. All RT_STAP algorithms apply nonadaptive temporal filtering (Doppler processing) on the data from each array element prior to adaptive processing. This operation transforms the space-time snapshot into a snapshot of Doppler bin and element data (represented by the upper-right corner of Figure 7). STAP algorithms that operate on a subset of this data are termed *element-space post-Doppler* algorithms, as the adaptation occurs after Doppler processing, and require solving a separate adaptive problem for each Doppler bin.

The next three subsections detail the three different types of adaptive processing presented in the RT_STAP package: post-Doppler adaptive DPCA, first-order Doppler-factored STAP, and high-order Doppler-factored STAP. The algorithms compute a set of adaptive weights using an approach that involves a matrix factorization called the QR decomposition [Golub96:223]. The computational complexity of the different approaches is governed by the size of the QR decomposition (size of the base matrix) and the number of decompositions required [Cain97].

C.3.1 High-Order Doppler-Factored STAP

This kind of algorithm implements temporal adaptivity by combining multiple Doppler filters from each element. Each filter may be thought of as a different windowing of the pulses, so such an architecture is called multiwindow post-Doppler STAP. The architecture is composed of Doppler processing across all PRIs followed by adaptive filtering across sensors and adjacent Doppler bins. Adaptive filtering of the data uses simultaneous spatial and temporal degrees of freedom (DOF) in each specified Doppler bin. The spatial DOF are provided by the L array channels, while the temporal

DOF are provided by the Q adjacent Doppler bins centered about the specified Doppler bin. A graphical representation is provided in Figure C.4.

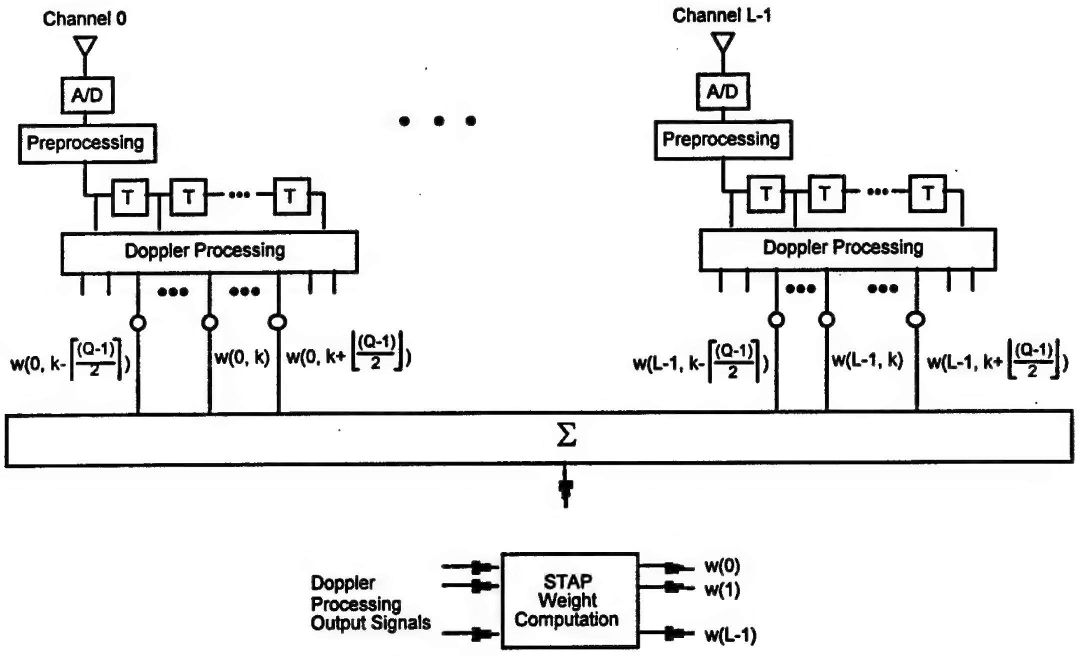

Channel 0
Channel L-1
A/D
Preprocessing
T
Doppler Processing
$w(0, k-\lceil\frac{(Q-1)}{2}\rceil)$
$w(0, k)$
$w(0, k+\lfloor\frac{(Q-1)}{2}\rfloor)$
$w(L-1, k-\lceil\frac{(Q-1)}{2}\rceil)$
$w(L-1, k)$
$w(L-1, k+\lfloor\frac{(Q-1)}{2}\rfloor)$
Σ
Doppler Processing Output Signals
STAP Weight Computation
w(0)
w(1)
w(L-1)

Figure C.4 – High-order Doppler-factored STAP.

The adaptive weights for a particular range cell r and Doppler bin k are computed from the second-order statistics [Reed74] of the space-time snapshot vector $\vec{x}(k,r)$ consisting of data samples across the L array channels and the Q adjacent Doppler bins $k_{\min}$ through $k_{\max}$ that are centered about Doppler bin k. For Q^{th} order Doppler-factored STAP, we define $k_{\min}$ to be $\bmod_K\left(k - \lfloor (Q-1)/2 \rfloor\right)$ and $k_{\max}$ to be $\bmod_K\left(k + \lceil (Q-1)/2 \rceil\right)$, where $\bmod_K(\cdot)$ is the modulo operator (e.g., $\bmod_K(-1) = K-1$ and $\bmod_K(K) = 0$). The $\hat{L} \times 1$ space-time snapshot vector is defined:

$$\vec{x}(k,r) = \left[x_5(0,k_{\min},r) \cdots x_5(L-1,k_{\min},r) \cdots x_5(0,k_{\max},r) \cdots x_5(L-1,k_{\max},r)\right]^T,$$

where $\hat{L} = L \cdot Q$ and $x_5(l,k,r)$ represents the data sample corresponding to the Doppler processing output for the rth range cell, kth Doppler bin, and lth channel. A graphical representation of $\vec{x}(k,r)$ is sketched in Figure C.5.

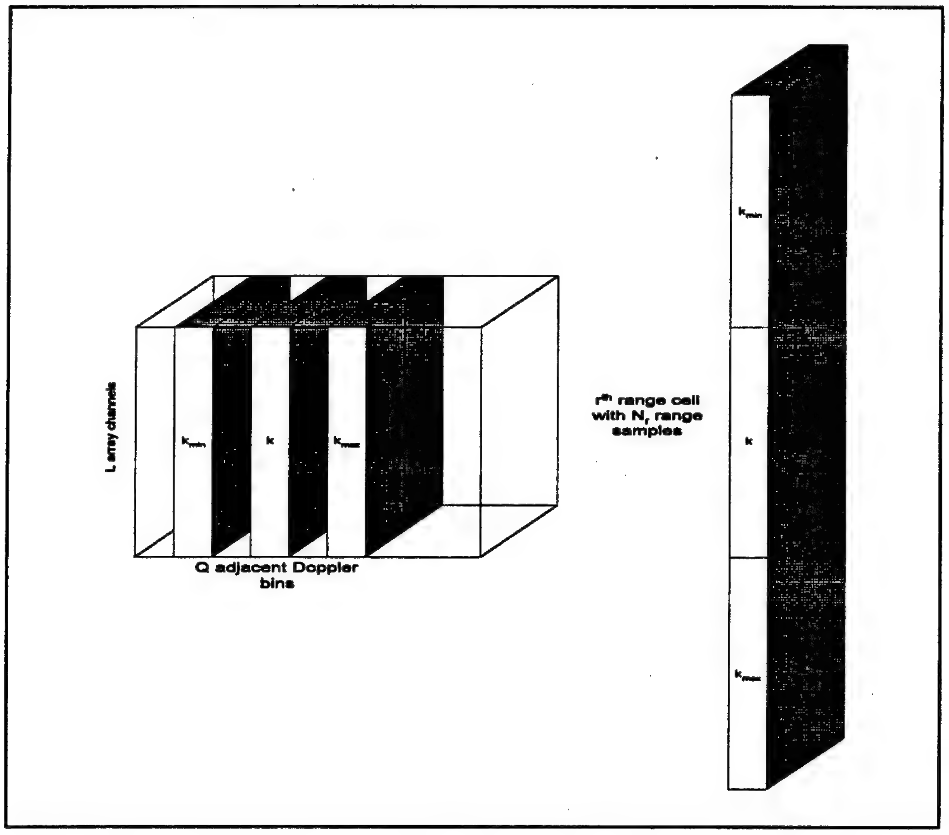


Figure C.5 – Graphical representation of $\vec{x}(k,r)$ for k = 3.

Given this definition of $\vec{x}(k,r)$, the $\hat{L} \times 1$ column vector of space-time adaptive weights $\vec{w}(k,r)$ is defined as

$$\vec{w}(k,r) = \left[w(0,k_{\min},r) \cdots\ w(L-1,k_{\min},r) \cdots w(0,k_{\max},r) \cdots\ w(L-1,k_{\max},r)\right]^{\mathrm{T}},$$

where $w(l,k,r)$ represents the adaptive weight to be applied to the data sample corresponding to the r^{th} range cell, k^{th} Doppler bin, and l^{th} channel. The output of the adaptive processing for the k^{th} Doppler bin and r^{th} range cell is:

$$x_6(k,r) = \vec{w}^{\mathrm{H}}(k,r)\vec{x}(k,r).$$

The outputs $x_6(k,r)$ for $r = 0,\ldots,N_{\mathrm{D}} - 1$ and $k = 0,\ldots,K-1$ result in the range-Doppler map following adaptive processing. Detection algorithms can then be applied to the result to locate targets in range and Doppler.

By defining the space-time covariance matrix to be:

$$\vec{\Psi}(k,r) = \mathrm{E}\{\vec{x}(k,r)\vec{x}^{\mathrm{H}}(k,r)\},$$

where $\mathrm{E}\{\cdot\}$ is the expectation operator, the adaptive weights are obtained by solving the following system of linear equations:

$$\vec{\Psi}(k,r)\,\vec{w}(k,r) = \gamma\,\vec{s}, \qquad \text{(C-1)}$$

where γ is a scale factor chosen such that $\vec{w}^{\mathrm{H}}(k,r)\,\vec{s}$ equals unity. The $\hat{L} \times 1$ column vector $\vec{s}$ corresponds to the target space-time steering vector defined to be:

$$\vec{s} = [s(0,0) \cdots s(L-1,0) \cdots s(0,Q-1) \cdots s(L-1,Q-1)]^{\mathrm{T}},$$

where,

$$s(l,q) = \hat{s}(l)g(q + \operatorname{mod}_K(-\lfloor (Q-1)/2 \rfloor)),$$

$q = 0,\ldots,Q-1$, $l = 0,\ldots,L-1$, $\hat{s}(l)$ corresponds to the target spatial steering vector at the l-th channel,

$$g(k) = \sum_{p=0}^{P-1} d(p)e^{j(2\pi/K)pk},$$

and $d(\cdot)$ represents the real-valued Doppler window. The space-time steering vector is normalized such that $\vec{s}^{\mathrm{H}}\vec{s}$ equals unity.

The higher-order Doppler-factored STAP algorithm clearly depends upon knowledge of the space-time covariance matrix. For practical applications, this matrix is unknown and must be estimated from the data samples. In general, an estimate of the covariance matrix is computed by averaging over snapshot vectors from adjacent range cells, as discussed in Chapter II.

The training strategy involves dividing the range cells into M non-overlapping blocks containing N_{R} contiguous range samples, where $M = N_{\mathrm{D}} / N_{\mathrm{R}}$ and where N_{R} is selected so that the ratio is an integer. The covariance matrix for the k th Doppler bin and m th block of contiguous range cells is computed by averaging over the outer product of the snapshot vectors. That is:

$$\vec{\Psi}(k,m) = \frac{1}{N_{\mathrm{R}}} \sum_{r=r_1}^{r_1+N_{\mathrm{R}}-1} \vec{x}(k,r)\vec{x}^{\mathrm{H}}(k,r),$$

where $\vec{\Psi}(k,m)$ is the estimate of the covariance matrix, $r_1 = mN_{\mathrm{R}}$, $m = 0,\ldots,M-1$, and $k = 0,\ldots,K-1$. The estimate is used in place of the covariance matrix to compute the adaptive weight vector that is applied to all the data snapshots comprising the m th block of range cells for the k th Doppler bin.

If the $\hat{L} \times N_{\mathrm{R}}$ space-time data matrix, $\vec{X}(k,m)$, is defined to be:

$$\vec{X}(k,m) = \left[\vec{x}(k,mN_{\mathrm{R}}) \cdots \vec{x}(k,(m+1)N_{\mathrm{R}}-1)\right], \tag{C-2}$$

then

$$\vec{\Psi}(k,m) = \frac{1}{N_{\mathrm{R}}} \vec{X}(k,m)\vec{X}^{\mathrm{H}}(k,m).$$

Equation (C-1) then becomes:

$$\vec{X}(k,m)\vec{X}^{\mathrm{H}}(k,m)\underline{\vec{w}}(k,m) = \underline{\gamma}\, N_{\mathrm{R}}\vec{s}\,,$$

where $\underline{\vec{w}}(k,m)$ corresponds to the weight vector applicable to all range cells in the m th block and is computed using the data matrix $\vec{X}(k,m)$. The scale factor $\underline{\gamma}$ is chosen such that $\underline{\vec{w}}^{\mathrm{H}}(k,m)\vec{s}$ equals unity.

The number of range samples used to estimate the covariance matrix should be at least twice the number of DOF (i.e., $\hat{L}$) [Cain97][McMahon96]. The training strategy described relies upon the knowledge of accurate spatial steering vectors to prevent significant target signal cancellation.

C.3.1.1 Weights Computation

The weight vector is computed by first performing a QR-decomposition on the full column-rank space-time data matrix $\vec{X}^{\mathrm{T}}(k,m)$ defined in equation (C-2). The transpose $\vec{X}^{\mathrm{T}}(k,m)$ is used to conform with the least-squares convention of having an over-determined system with more rows (N_{R}) than columns ($\hat{L}$). The QR-decomposition produces an $N_{\mathrm{R}} \times N_{\mathrm{R}}$ unitary matrix, $\vec{Q}$, and an $N_{\mathrm{R}} \times \hat{L}$ upper triangular matrix, $\vec{R}$, such that $\vec{X}^{\mathrm{T}}(k,m) = \vec{Q}\,\vec{R}$. The matrix $\vec{R}$ can be written as $\begin{bmatrix}\vec{R}_1^{\mathrm{T}} & \vec{0}\end{bmatrix}^{\mathrm{T}}$, where $\vec{R}_1$ is a $\hat{L} \times \hat{L}$ full rank upper triangular matrix. The matrix product $\vec{X}(k,m)\vec{X}^{\mathrm{H}}(k,m)$ decomposes to

$$\vec{X}(k,m)\vec{X}^{\mathrm{H}}(k,m) = \vec{R}^{\mathrm{T}}\,\vec{Q}^{\mathrm{T}}\,\vec{Q}^{*}\,\vec{R}^{*} = \vec{R}_1^{\mathrm{T}}\,\vec{R}_1^{*}\,,$$

where $\vec{Q}^{\mathrm{T}}\vec{Q}^{*} = \left(\vec{Q}^{\mathrm{H}}\vec{Q}\right)^{*} = I$. Since the matrix $\vec{Q}$ is not involved in the weight computation, it is not necessary to explicitly compute this matrix during the QR-decomposition process.

The *modified Gram-Schmidt* method is used for computing QR decomposition [Golub96:231], and $8 \cdot N_{\mathrm{R}} \cdot [L \cdot Q]^2$ floating-point operations are required to implement a single QR-decomposition corresponding to the m th block of range cells for the k th Doppler bin. A total of $K \cdot M \cdot \left(8 \cdot N_{\mathrm{R}} \cdot [L \cdot Q]^2\right)$ floating-point operations are required to implement all the QR-decompositions involved in the application of Q th-order Doppler-factored STAP in each Doppler bin and each block of range cells in the data cube.

Following the QR-decomposition, forward elimination and backward substitution are performed to solve for the adaptive weights [Golub96:87]. The weight vector can be computed by first solving for the vector $\vec{p}$ in the expression:

$$\vec{R}_1^{\mathrm{T}}\,\vec{p} = N_{\mathrm{R}}\,\vec{s}$$

using forward elimination. The weight vector is determined by solving the expression:

$$\vec{R}_1^{*}\,\vec{r} = \vec{p}$$

using backward substitution. The final weight vector $\vec{\underline{w}}(k,m)$ is equivalent to $\underline{\gamma}\vec{r}$ where $\underline{\gamma}$ is selected so that $\vec{\underline{w}}(k,m)^{\mathrm{H}}\vec{s} = 1$. As a result, $\underline{\gamma} = \left(\vec{r}^{\mathrm{H}}\vec{s}\right)^{-1}$.

Forward elimination and back substitution each require $4 \cdot [L \cdot Q]^2$ floating-point operations to implement. Clearly, implementation of the QR-decomposition dominates the computational complexity of the STAP weight computation process. Adaptive weights must be computed in each Doppler bin and each block of range cells. Consequently, the total number of computations required to solve for the adaptive weights for the entire data cube is dominated by $K \cdot M \cdot \left(8 \cdot [L \cdot Q]^2 \cdot (N_\mathrm{R} + 1)\right)$.

C.3.1.2 Weights Application

If we let $\underline{\vec{w}}(k, m)$ represent the adaptive weight vector to be applied to data corresponding to the k-th Doppler bin and the m-th block of range cells, then the N_R outputs of the STAP algorithm are given by the product of the data matrix and the weight vector:

$$\underline{\vec{w}}^\mathrm{H}(k, m)\vec{X}(k, m).$$

This process requires $8 \cdot L \cdot Q \cdot N_\mathrm{R}$ floating-point operations to implement, and it must be repeated in each Doppler bin and each block of range cells in the data cube. The total number of floating-point operations required to apply the adaptive weights to the data cube is $K \cdot M \cdot (8 \cdot L \cdot Q \cdot N_R)$. The Section C.3.1 evaluated the computational complexity of the high-order Doppler-factored STAP, and the results of the analysis are summarized in Table C.2.

Function	Operation Count
Doppler Processing	$L \cdot N_\mathrm{D} \cdot (5 \cdot K \cdot \log_2 K + 2 \cdot P)$
Weights Computation	$K \cdot M \cdot \left(8 \cdot [L \cdot Q]^2 \cdot (N_\mathrm{R} + 1)\right)$
Weights Application	$K \cdot M \cdot (8 \cdot L \cdot Q \cdot N_R)$

Table C.2 - Higher-Order Doppler-Factored STAP computation counts.

The definitions of the variables used in the Table C.2 are:

L:	Number of channels
P:	Number of pulses per Doppler processing block
N_{D}:	Number of samples per pulse after decimation
K:	Doppler FFT size (power of 2)
M:	Number of independent non-overlapping blocks $N_{\mathrm{D}} / N_{\mathrm{R}}$ of contiguous range samples used to calculate the adaptive weights
N_{R}:	Number of contiguous range cells per weight computation
Q:	Processing order.

C.3.2 First-Order Doppler-Factored STAP

The first-order Doppler-factored STAP algorithm is one of the simplest post-Doppler STAP techniques known for clutter and interference suppression. This algorithm utilizes a single Doppler filter bank on each element. Adaptive spatial beamforming is then performed separately within each Doppler bin. Strictly speaking, this approach is not really a space-time adaptive algorithm, as the adaptive weights are spatial only (i.e., just a single DOF), but it sure provides a significant reduction in dimensionality for the problem. The architecture of the first-order Doppler-factored STAP algorithm corresponds to the processing of a *single Doppler bin* in the higher-order Doppler-factored architecture shown in Figure C.4.

C.3.3 Post-Doppler Adaptive DPCA

The post-Doppler adaptive DPCA algorithm was one of the first techniques developed to address the issue of Doppler-spread clutter in airborne radar [Skolnik90]. It employs simultaneous spatial and temporal filtering, as shown in Figure C.6, to suppress the sidelobe clutter competing with the target. The basic concept is to adjust the radar

PRI such that the motion of the aircraft platform causes channel (i.e., phase center) 0 to move exactly into the spatial position of channel 1 after a single PRI (i.e., pulse). The processing differs from first-order Doppler-factored STAP only in that an additional time delay is included in channel one. This added time delay changes the Doppler processing implemented in the two channels so that the K complex data samples after Doppler processing are:

$$x_5(0,k,r) = \sum_{p=0}^{P-1} d(p) x_4(0,p,r) e^{j(2\pi/K)pk}$$

and

$$x_5(1,k,r) = \sum_{p=0}^{P-1} d(p) x_4(1,p+1,r) e^{j(2\pi/K)pk},$$

for $r = 0,\ldots,N_D - 1$ and $k = 0,\ldots,K-1$. In other words, channel 0 performs Doppler processing on pulses 0 through $P-1$, while channel 1 processes pulses 1 through P.

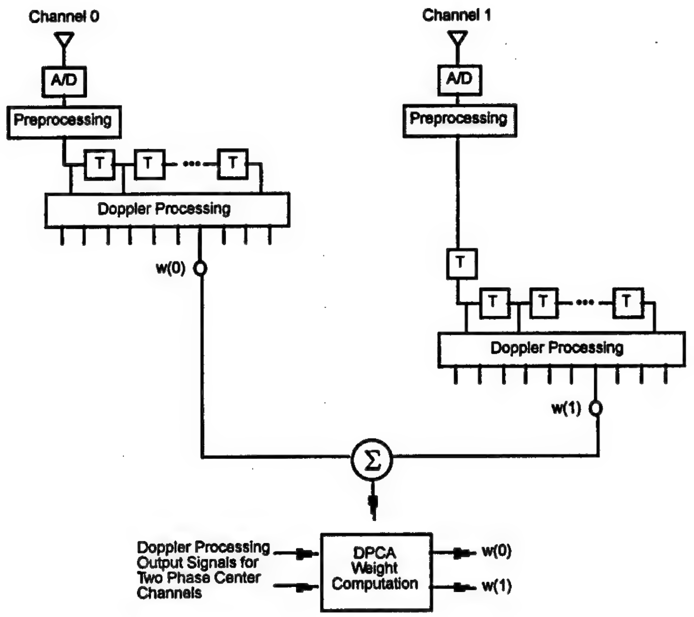


Figure C.6 - Post-Doppler Adaptive DPCA.

C.4 Distributed Cornerturns

Cornerturning is the name given for transposing a matrix in some signal processing applications [Games96]. To meet stringent real-time latency constraints the rows or columns of the matrices involved are often distributed across many processing nodes.

The cornerturn data redistribution operation usually consists of three phases as shown for the case of four processing nodes in Figure C.7. First, there are memory transposes, or local corner turns, at the nodes that take each node's portion of the matrix stored by rows, say, and restores it by columns. There are no MPI operations involved in this first step. Then, each portion is packaged into large messages - using *MPI_Pack* - for more efficient transmission. The second phase corresponds to the data distribution phase shown in step 2 of Figure C.7. In this All-to-All communication step each node communicates some portion of its data to every other node. Finally, a second memory copy at the processing nodes is needed to unpack the messages and to store the result by columns for subsequent processing.

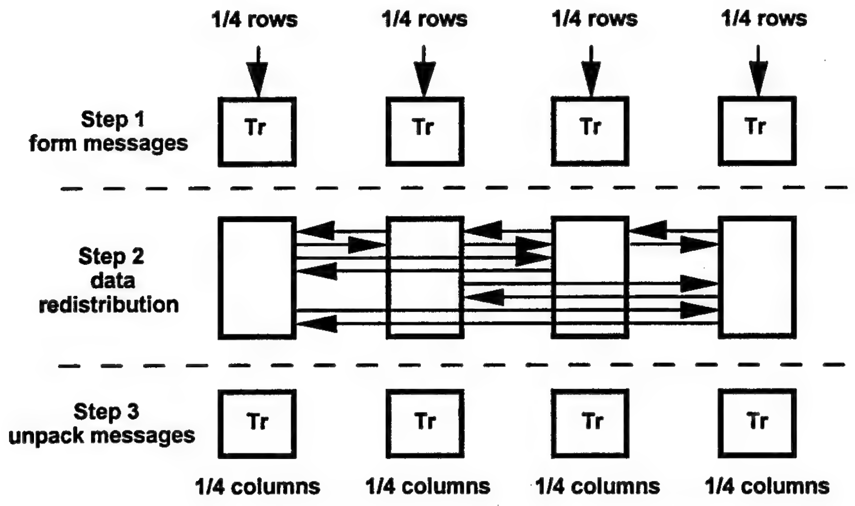


Figure C.7 – Cornerturning.

All data reorganizations executed during run time in the RT_STAP programs are based on the distributed cornerturns described above.

The implementation is *in-place*, i.e., steps 1, 2, and 3 take place on the same group of *N* processing nodes, instead of *pipelined*, where step 1 occurs on a group of source nodes, the step 2 data redistribution occurs over the network connecting the source nodes and a group of sink nodes, and step 3 occurs on the sink nodes. This corresponds to the case that the source nodes process the rows of the matrix and the sink nodes process the columns. Figure C.8 gives a graphical representation of the difference. In this sense, MPI collective operations currently do not support pipelined implementations.

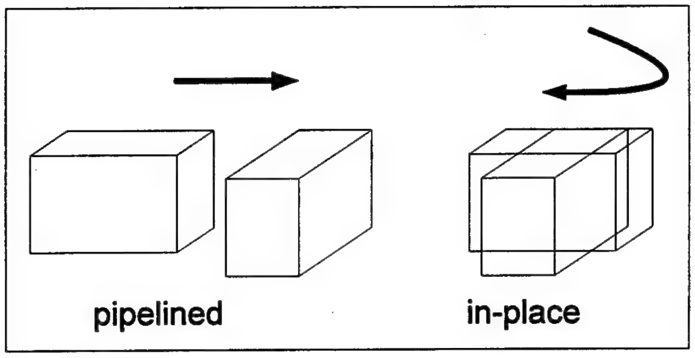


Figure C.8 – Graphical representation of a cornerturn operation. The pipeline type requires a different set of processors to enable communication.

Data reorganization operations occur twice in the RT_STAP implementation: (a) after preprocessing; (b) after Doppler processing. Picking the high-order post-Doppler STAP as an illustrative example, we see that the second cornerturn change positions on all the axis: 22 channels, 480 ranges, and 64 Doppler bins are reorganized as 64 Doppler bins, 22 channels, and 480 ranges. This turn is realized by considering the cube across processes as a matrix distributed across processes. There are three basic stages to realizing this cubic turn: combine the source cube's first two dimensions into a unique dimension, combine the destination cube's last two dimensions into a unique dimension, and using these calculated matrices distributions, turn the cube as a matrix. That is,

Source to matrix : ijk → hk

Matrix cornerturn to destination : hk → kh

Destination matrix to cube : kh → kij

There is no actual data movement by translating a cube into a matrix. It is just a different way of viewing the data. The data at any point can be viewed as either a matrix or cube. After the adaptive processing, a reduction on dimensionality is achieved, and the final organization is two-dimensional: 64 Doppler bins by 480 ranges.

Bibliography

[Anderson95] T. Anderson, D. Culler and D. Patterson, "A Case for NOW," IEEE Micro, vol. 15, no. 1 pp. 54-64, Feb. 1995.

[Antonik97] P. Antonik and others, "Knowledge-Based STAP," Proceedings of the IEEE National Radar Conference, pp. 372-376, 1997.

[Barr95] R. Barr and others, "Designing and Reporting on Computational Experiments with Heuristic Methods," Journal of Heuristics, no. 1 pp. 9-32, Kluwer Academic Publishers, 1995.

[Brennan73] L. Brennan and I. Reed, "Theory of Adaptive Radar," IEEE Transactions in Aerospace and Electronic Systems, vol. 9, no. 2, pp. 237-252, 1973.

[Brian98] B. Sroka and K. Cain, "Portable Software Library Optimization," Technical Report MTR 98B0000037, MITRE Corporation, Bedford, Mass: Feb. 1998.

[Briggs95] W. Briggs and V. Henson, The DFT – An Owner's Manual for the Discrete Fourier Transform. Philadelphia: SIAM, 1995.

[Browne98] S. Browne and others, "Review of Performance Analysis Tools for MPI Parallel Programs," WWWeb http://www.cs.utk.edu/~browne/perftools-review/, Dec. 1998.

[Cain97] K. Cain and others, "RT_STAP: Real-Time Space-Time Adaptive Processing Benchmark," Technical Report MTR 96B0000021, MITRE Corporation, Bedford, Mass.: Feb. 1997.

[Censor97] Y. Censor and S. A. Zenios, Parallel Optimization – Theory, Algorithms, and Applications. New York: Oxford University Press, 1997.

[Chorafas97] D. Chorafas, Network Computers versus High Performance Computers. London: Cassell Plc., 1997.

[Chouldhary97] A. Chouldhary and others, "Design, Implementation and Evaluation of Parallel Pipelined STAP on Parallel Computers," New York: 1997.

[Cooley65] J. Cooley and J. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, vol.19, pp. 297-301, 1965.

[CREST96] Common Research Environment for STAP Processing. Maui High Performance Computing Center at http://wwwcrest.mhpcc.edu/, 1996.

[Culler96] D. E. Culler and others, "Assessing Fast Network Interfaces," IEEE Micro, pp. 35-43, Feb. 1996.

[Culler97] D. E. Culler and others, "Parallel Computing on the Berkeley NOW," WWWeb http://www.cs.berkeley.edu/Research/Projects, 1997.

[Culler98] D. E. Culler and others, Parallel Computer Architecture: A Hardware/Software Approach. San Francisco: Morgan Kaufmann Publishers, 1998.

[Cuthbert80] D. Cuthbert and F. Wood, Fitting Equations to Data. New York: John Wiley & Sons, 2nd ed., 1980.

[Davenport87] W. B. Davenport Jr., Probability and Random Processes – An Introduction for Applied Scientists and Engineers. McGraw-Hill, 1987.

[Devore95] J. L. Devore, Probability and Statistics for Engineering and the Sciences. Duxbury Press, 4th ed., 1995.

[DeBar98] J. DeBardelaben, "Cost Modeling for Embedded Digital Systems Design," RASSP Education and Facilitation Program – Module 57, WWWeb http://www.cedcc.psu.edu/ee497i/rassp_57/ , Jun 1998.

[Dongarra92] J. Dongarra and others, "LAPACK Working Note 41 – Installation Guide for LAPACK," UT, CS-92-151, Feb. 1992.

[Dongarra94] J. Dongarra and S. Ostrouchov, "LAPACK Working Note 81 – Quick Installation Guide for LAPACK on Unix Systems," Knoxville: version 2.0, Sep. 1994.

[Dongarra98] J. Dongarra and others, Numerical Linear Algebra for High-Performance Computers. Philadelphia: SIAM, 1998

[Eaves87] J. L. Eaves and E. K. Reedy, Principles of Modern Radar. New York: Van Nostrand Reinhold, 1987.

[Flynn72] M. Flynn, "Some Computer Organizations and their Effectiveness," IEEE Transactions on Computers, vol. C-21, no. 9, pp. 948-960, Sep. 1972.

[Games96] R. A. Games, "Benchmarkimg Methodology for Real-Time Embedded Scalable High Performance Computing," Technical Report MTR 96B0000010, MITRE Corporation, Bedford, Mass.: Mar. 1996.

[Games98] R. A Games, "Common Processors for Signal and Image Processing: Toward a Real-Time Embedded Common Operating Environment (RTE-COE)," Technical Report MTR 980000020, MITRE Corporation, Bedford, Mass.: Apr. 1998.

[Goodman63] N. R. Goodman, "Statistical Analysis Based on Certain Multivariate Complex Gaussian Distribution," Annals of Mathematical Statistics, vol. 34, pp. 152-177, Mar. 1963.

[Golub96] G. H. Golub and C. F. Van Loan, Matrix Computations. London: The John Hopkins Press Ltd., 3rd. ed., 1996.

[Grama93] A. Grama and others, "Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures," IEEE Parallel and Distributed Technology, vol. 1, no. 3, pp.12-21, 1993.

[Gustafson88] J. Gustafson, "Reevaluating Amdahl's Law," Communications of the ACM, vol.31 no. 5, pp. 532-533, 1988.

[Gustafson92] D. Gustafson, "The Scalable Coherent Interface and Related Standards Projects," IEEE Micro, pp. 10-12, Feb. 1992.

[Haykin91] S. Haykin, Adaptive Filter Theory. New Jersey: Prentice Hall, 1991.

[Hebert98] L. Hebert and others, "MPI for Windows NT: Two Generations of Implementations and Experience with the Message Passing Interface for Clusters and SMP Environments," Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pp. 309-316, 1998.

[Hennessy96] J. Hennessy and D. Patterson, Computer Architecture – A Quantitative Approach. San Francisco: Morgan Kaufmann Publishers, 1996.

[Higham96] N. Higham, Accuracy and Stability of Numerical Algorithms. Philadelphia: SIAM, 1996.

[Hill95] M. Hill and D Wood, "Cost-Effective Parallel Computing," IEEE Computer, 28 (2), pp.69-72, 1995.

[Hockney88] R.W. Hockney, C. Jesshope, Parallel Computers II: Architecture, Programming and Algorithms. Adam Hilger, Ltd., Bristol, United Kingdom, 1988.

[Hwan96] K. Hwang and Z. Xu, "Scalable Parallel Computers for Real-Time Signal Processing," IEEE Signal Processing Magazine, pp. 50-66, Jul 1996.

[Hwang96] K. Hwang and others, "Benchmark Evaluation of the IBM SP2 for Parallel Signal Processing," IEEE Transactions on Parallel and Distributed Systems, vol. 7, no. 5, pp.522-535, May 1996.

[Hwang98] K. Hwang and Z. Xu, Scalable Parallel Computing – Technology, Architecture, Programming. WCB/McGraw-Hill, 1998.

[IBM98] IBM specifications for the series SP. WWWeb http://www.rs6000.ibm.com, 1998.

[Jain91] R. Jain, The Art of Computer Systems Performance Analysis – Techniques for Experimental Design, Measurement, Simulation, and Modeling. New York: John Wiley & Sons, Inc., 1991.

[James95] J. Anderson, "Projecting RASSP Benefits," Program for Rapid Prototyping of Application Specific Signal Processors (RASSP) WWWeb http://llex.ll.mit.edu/llrassp, 1998.

[Kenneth97] K. Cain Jr. and B. Sroka, "Experiences in Porting and Existing Application to Use the VSIP API," WWWeb http://www.mitre.org, Jul. 1997.

[Kumar94] V. Kumar and others, Introduction to Parallel Computing – Design and Analysis of Algorithms. Redwood City: The Benjamin/Cummings Publishing Company, 1994.

[Lamont97] G. B. Lamont and D. M. Gallagher, "Scalable Distributed Multi-Dimensional FFT Algorithm Design, Implementation, and Analysis," Interim Report 2, Air Force Institute of Technology, Summer/Fall 1997.

[Lauria98] M. Lauria and others, "Cross Platform Analysis of Fast Messages for Myrinet," Lecture Notes in Computer Science 1362, pp.217-231, Springer-Verlag, 1998.

[Lawson79] C. Lawson and others, "Basic Linear Algebra Subprograms for FORTRAN Usage," ACM Transactions on Mathematical Software, vol. 5, pp. 308-329, 1979.

[Lebak96] J. M. Lebak and others, "Toward a Portable Parallel Library for Space-Time Adaptive Methods," CTC96TR242 6/96, Rome Laboratories, New York: 1996.

[Lenoski92] D. Lenoski and others, "The Stanford DASH Multiprocessor," Computer, pp. 63-79, Mar. 1992.

[Linder97] M. Linderman and R. Linderman, "Real-Time STAP Demonstration on an Embedded High Performance Computer," Proceedings of the IEEE National Radar Conference, pp. 54-59, 1997.

[Liu94] M. Liu and others, "Distributed Network Computing over Local ATM Networks," Proceedings of Supercomputing '94, 1994.

[Lockheed98] Lockheed Martin Corporation. WWWeb, http://www.lmco.com/, 1998

[Maill93] R. J. Mailloux, Phased Array Antenna Handbook. Boston, MA: Artech House, 1993.

[McMahon96] J. O. McMahon and K. Teitelbaum, "Space-Time Adaptive Processing on the Mesh Synchronous Processor," 10th International Parallel processing Symposium. Honolulu: pp. 734-740, 1996.

[MIT/LL94] MIT Lincoln Laboratory, "STAP Processor Benchmarks," Lexington: Feb. 1994.

[MNR98] Mercury News Release, WWWeb, http://www.mc.com/press/97_9_8_ng_stap.html, Aug. 1998.

[Morris88] G. V. Morris, Airborne Pulsed Doppler Radar, Norwood, MA: Artech House, 1988.

[MPI] MPI. The Message Interface Standard. WWWeb http://www.mcs.anl.gov/mpi/index.html.

[Napear95] S. Napear and D. Nadeau, "Visualization of STAP," presented at the Third Annual Adaptive Sensor Array Workshop, MIT Lincoln laboratory, Mar. 1995.

[Nupairoj94] N. Nupairoj and L. Ni, "Performance Evaluation of Some MPI Implementations on Workstation Clusters," Proceedings of the Scalable Parallel Libraries Conference, Oct. 1994.

[OST93] Office of Science and Technology Policy. "Grand Challenges 1993: High Performance Computing and Communications, A Report by the Committee on Physical, Mathematical, and Engineering Sciences," 1993.

[Pacheco97] P. Pacheco, Parallel programming with MPI. San Franciso: Morgan Kaufmann, 1997.

[Pakin97] S. Pakin and others, "Fast Messages: Efficient and Portable Communication for Workstation Clusters and Massively-Parallel Processors," IEEE Concurrency, vol. 5, No. 2, pp. 60-73, 1997.

[Papoulis91] A. Papoulis, Probability, Random Variables, and Stochastic Processes. 3rd. ed., McGraw-Hill, 1991.

[Pasquale94] B. K. Pasquale and G. C. Polyzos, "Dynamic I/O Characterization of I/O Intensive Scientific Applications," Proceedings of Supercomputer 1994, pp. 660-669, Nov. 1994.

[Patterson98] D. Patterson and J. Hennessy, Computer Organization & Design – A Hardware/Software Interface. San Francisco: Morgan Kaufmann, 2 ed., 1998.

[Pfister98] G. F. Pfister, In Search of Clusters. 2 ed., Upper Saddle River, NJ: Prentice Hall Inc., 1998.

[Rabaey98] J. M. Rabaey, "VLSI Implementation Fuels the Signal Processing Revolution," IEEE Signal Processing Magazine, Jan. 1998.

[Reed74] I. S. Reed and others, "Rapid Convergence Rate in Adaptive Arrays," IEEE Transactions on Aerospace and Electronic Systems, AES-10, No. 6, Nov. 1974.

[Ridge97] D. Ridge and others, "Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs," Proceedings, IEEE Aerospace, WWWeb http://cesdis.gsfc.nasa.gov/beowulf/papers/AA97/final.ps, 1997

[Rihac69] A. W. Rihaczek, Principles of High-Resolution Radar. New York: McGraw-Hill, 1969.

[Rothnie92] J. Rothnie, "Overview of the KSR1 Computer System," Technical Report TR2020001, KSR Corporation, Cambridge, MA, Mar. 1992.

[Rowe96] P. K. Rowe, "COTS Radar and Sonar Systems Solutions," Multiprocessor Toolsmiths Inc., Kanata, ON Canada, 1996.

[Samson96] J. R. Samson and others, "STAP Performance on a Paragon/Touchstone System," Proceedings of the IEEE National Radar Conference, Ann Harbor, Michigan, pp. 315-320, May 1996.

[Scott98] S. Berger and B. Welsh, "Selecting a Reduced-Rank Transformation for STAP – a Direct Form Perspective," Proceedings of the IEEE Radar Conference, Dallas, Texas, pp. 177-182, May 1998.

[Skolnik62] M. I. Skolnik, Introduction to Radar Systems. New York: McGraw-Hill, 1962.

[Skolnik90] M. I. Skolnik, Radar Handbook. New York: McGraw-Hill, 2 ed., 1990.

[Smirni97] E. Smirni and D. A. Reed, "Workload Characterization of I/O Intensive Parallel Applications," Lecture Notes in Computer Science 1245, pp. 169-180, Heidelberg: Springer-Verlag, 1997.

[Sound98] V. Soundararajan and others, "Flexible use of Memory for Replication/Migration in Cache-Coherent DSM Multiprocessors," 25th Annual International Symposium on Computer Architecture, pp. 342-355, Barcelona: Jun. 1998.

[Staud90] F. M. Stauder, "Airborne MTI," Radar Handbook, Chapter 16, editor M. I. Skolnik. New York: McGraw-Hill, 1990.

[Steen98] A. Steen and J. Dongarra, "Overview of Recent Supercomputers," WWWeb http://www.netlib.org/utk/papers/advanced-computers/overview98.html, Feb 1998.

[Sterling95] T. Sterling and others, "Beowulf: A Parallel Workstation for Scientific Computing," Proceedings, International Conference on Parallel Processing, WWWeb http://www.beowulf.org/papers/IPPC95/final.ps, 1995.

[Stimson98] G. W. Stimson, Introduction to Airborne Radar. New jersey: SciTech Publishing Inc., 2nd. ed., 1998.

[Sun94] X. Sun and D. Rover, "Scalability of Parallel Algorithm-Machine Combinations," Parallel and Distributed Systems, vol. 5, no. 6, pp. 599-613, 1994.

[Swarz82] P. Swarztrauber, "Vectorizing the FFTs," in Parallel Computations, pp.51-83. New York: Academic Press, 1982.

[Tanaka98] Y. Tanaka and others, "Performance Improvement by Overlapping Computation and Communication on SMP Clusters," Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pp 275-282, 1998.

[Taylor48] D. Taylor and C. H. Westcott, Principles of Radar. London: Cambridge University Press, 1948.

[Tex98] Texas Tech University, HPCL, WWWeb, http://hpcl.cs.ttu.edu/darpa.html, Aug 1998.

[Toomay89] J. C. Toomay, Radar Principles for the Non-Specialist. New York: Van Nostrand Reinhold, 1989.

[Tufte83] E. Tufte, The Visual Display of Quantitative Information. Cheshire: Graphic Press, 1983.

[UIUC98] University of Illinois at urbana-Champaign, "High-performance Supercomputing at Mail-Order Prices," WWWeb http://access.ncsa.uiuc.edu/CoverStories/SuperCluster/super.html., Jul 1998.

[VSIPL98] The Vector Signal Image Processing Library Forum, WWWeb http://www.vsip.org.

[Ward94] J. Ward, "Space-Time Adaptive Processing for Airborne Radar," Technical Report 1015, MIT Lincoln Laboratory, DTIC no. AD-A293032: 1994.

[Welsh98] T. Hale and B. Welsh, "Secondary Data Support in Space-Time Adaptive Processing," Proceedings of the IEEE Radar Conference, Dallas, Texas, pp. 183-188, May 1998.

[Zelkowitz98] M. Zelkowitz and D. Wallace, "Experimental Models for Validating Technology," Computer, Vol. 31, no. 5, IEEE, May 1998.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 15 June 1999	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE PARALLEL DIGITAL SIGNAL PROCESSING ON A NETWORK OF PERSONAL COMPUTERS. CASE STUDY: SPACE-TIME ADAPTIVE PROCESSING	5. FUNDING NUMBERS
6. AUTHOR(S) Luiz Fernando Silva, Captain, Brazilian Air Force	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology 2950 P Street, Bldg 640 WPAFB OH 45433-7765	8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/99J-01
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Brazilian Ministry of Aeronautics Esplanada dos Ministerios Brasilia - Distrito Federal Brasil	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES Dr. Gary B. Lamont COMM: (937) 255-3636 x4718 DSN: 785-3636 x4718	
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This work evaluates the capabilities and the performance of the Air force Institute of Technology's Pile-of-PCs for parallel digital signal processing using space-time adaptive processing (STAP) under the Linux OS. The MITRE RT_STAP Benchmark version 1.1 is ported and executed on it, as well as on a cluster of six Sun SPARC workstations connected by a Myrinet network (the AFIT NOW), and on a IBM SP for comparison. Modifications to the RT_STAP Benchmark source code are performed to accomodate the BLAS routines obtained from the US Department of Energy's Accelerated Strategic Computing Initiative, and the FFTPACK from the Netlib Repository, allowing improvements in the sustained Gflops/sec rates. However, the Pile-of-PCs also reveals limited scalability as a result of severe communication overheads imposed by RT_STAP cornerturn operations. Analysis of experimental data indicates that the PC Cluster outperforms AFIT NOW but needs interconnection network improvements to be globally competitive to multicomputers such as the IBM SP.

14. SUBJECT TERMS CLUSTER OF PCs, PARALLEL DIGITAL SIGNAL PROCESSING, STAP, REAL-TIME BENCHMARKING, LINUX	15. NUMBER OF PAGES 187
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL